1-1-2013

Combinatorial Auction-Based Virtual Machine Provisioning And Allocation In Clouds

Sharrukh Zaman
Wayne State University,

COMBINATORIAL AUCTION-BASED VIRTUAL MACHINE PROVISIONING AND ALLOCATION IN CLOUDS

by

SHARRUKH ZAMAN

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2013

MAJOR: COMPUTER SCIENCE

Approved by:

_____

Advisor Date

_____

_____

_____

DEDICATION

To my parents, who always sacrifice everything for their children's success.

ACKNOWLEDGMENTS

First, I would like to thank my advisor Dr. Daniel Grosu. His guidance, motivation, and support throughout the entire program is the most important factor that helped me keep in track. Besides his scholarly advice, I most appreciate his understanding of all human factors a Ph.D. student may face. He believes in his students, which is, in my opinion, the most precious asset a Ph.D. student needs to continue and finish this long journey. I am also indebted to my committee members Dr. Hongwei Zhang and Dr. Nathan Fisher for their support and encouragement since my qualifying exam. I repeatedly requested them for reference letters during my long-lasting job search, and every time they extended their help without hesitation. My external committee member Dr. Cheng-Zhong Xu was always flexible in scheduling my defense meetings despite his out of country travels. I also thankfully remember that he gave me the opportunity to review a paper in a renowned journal. I wish to thank the anonymous reviewers of our papers.

At this point I would like to express my gratitude to my parents Dr. Md. Badiuzzaman and Dr. Syeda Afifa Huda for always being there for me. They gladly accepted my long absence just because I will get this great opportunity to achieve the degree I dreamed of. My wife Masuma Khandaker married me when I left my job and was getting ready to start this long journey. She has been so patient and understanding with the limitations and crazy schedule of a graduate student, I will always wonder how she did this. I acknowledge my family members and friends who believed that I could finish this program. I remember Dr. M. Kaykobad of my undergraduate institution BUET at this moment. He always encouraged us to pursue higher degrees. He becomes really happy to learn about his former students' successes and tells the story to his current students.

The Department of Computer Science, despite many limitations, supports many Ph.D. students as GTAs and provides financial assistance for conference travels. The staff members are really good to make a foreign student feel home away from home. I also remember Rachel, Matt, and Deb, who left the department but were here most of the time I have been in the department. I was with a different group in my first year and at some point I thought I could not continue. Deb Mazur talked me out of that situation and encouraged me to find a group that would better suit my interest. This list would go on and I cannot even recall, let alone return the favor for every single contribution to my life and my success. I hope my degree will give me a greater opportunity to do good for the people who would come across my path and to the society at large.

Finally, this research was supported in part by National Science Foundation grants DGE-0654014 and CNS-1116787.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Cloud computing has revolutionized the computing world and will continue to do so for many years to come. The core technology behind cloud computing is virtualization. Large datacenters offer computing resources in terms of virtual machines where users remotely connect to perform their computing tasks or deploy their applications. Cloud services include virtual machines, virtual platforms, and cloud-based software. Big corporations that already possess large datacenters are able to generate additional revenues (e.g., Microsoft Windows Azure [42], Amazon EC2 [4]). Many new companies solely based on cloud computing products are being added to the market (e.g., Salesforce.com [54], Rackspace Hosting [51]). On the other hand, users, especially small and medium enterprises, now have a wide array of options for securing their computing resources - they can rely on cloud resources at any level (infrastructure, platform, or software). This enables users to balance their financial requirement for computing - they can replace the upfront cost of procurement and ongoing cost of in-house maintenance with utilizing cloud resources depending on their financial goals and limitations.

As an emerging field, cloud computing is the focus of many ongoing research. Among many tracks of cloud computing research, we identify that increasing the efficiency of allocation of the computing resources is a very important problem. As more users and providers enter the market, the more challenging the efficient allocation of computing resources will be. Efficiency of allocation, or *economic efficiency*, is achieved when it is ensured that the user who values an item the most, gets it. The value a user puts on an item can be known only when the user *bids* for that particular item, i.e., expresses her demand for that item and the amount she is willing to pay for it. However, to decide fairly who gets an item, it is required that all the users express their *true valuation* to the system. When it is guaranteed by a mechanism that the users can maximize their *utility* by telling the truth (i.e., reporting their true valuations for the requested resources), we say that the allocation

mechanism is *incentive compatible*.

The resource allocation model in cloud computing uses virtualization techniques. Cloud providers provision their computing resources into virtual machines (VMs) and users request the resources to accomplish their intended tasks. Current virtual machine allocation mechanisms employed by the cloud providers are mainly fixed-price based mechanisms. Economic theory shows that these mechanisms are not economically efficient. A fixed-price based mechanism usually follows a first-come fist-served approach that does not necessarily create incentives for the users to express their true valuation to the system. Regardless of how valuable a resource is to a user, she pays a fixed price to get access to that resource. On the other hand, a user who values a resource less than the published market rate cannot even participate in the process of requesting the resource for herself. As the cloud computing platform matures toward a popular and highly-accessed system, such issues will be of major concern for both the users and the providers.

An auction-based mechanism can provide fair allocation and economic provisioning of computing resources as virtual machines. A cloud provider needs to predict the demand of its resources in order to be able to provision them in a cost-effective and profitable way. An efficient mechanism can provide this information in real-time in order to enable the provider make their provisioning decision effectively. On the other hand, auction-based mechanisms can also help the cloud users plan proper level of resources for the tasks they want to execute. When a user knows that by asking for the resources they actually require with the price they want to really pay for that, her perceived utility from the allocation can be maximized, she will be able to realistically estimate her work performance and cost. Auctions are categorized based on different criteria. We determine that combinatorial auctions are best suited for allocating VM instances in clouds. Combinatorial auctions enable users to express their requirement when the items they require are complementary to one another. For example, a user hosting a web service may need three different servers to use as database server, application server, and web server. If she cannot acquire all three servers together, she would prefer not getting an allocation at all rather than acquiring a

subset of her requirement. Combinatorial auctions provide bidding protocols that allow complete expression of user requirements.

In this Ph.D. dissertation, we design combinatorial auction-based mechanisms to efficiently provision and allocate VM instances to cloud users, and determine their prices based on the market demand. We formulate different models for the problem of VM provisioning and allocation, and devise mechanisms to solve each of them. This will help a provider select a mechanism based on their particular needs. We have completed four research projects as part of this thesis. We designed two combinatorial auction-based mechanisms to allocate VM instances in a setting where they are assumed to be pre-provisioned. Later, we devised a dynamic mechanism that considers the entire pool of computing resources of a cloud provider as 'liquid' resources and provisions them dynamically into VM instances based on a combinatorial auction. The above two mechanism design problems are modeled as offline mechanism design problems. In an offline mechanism, the auction is run periodically and allocates resources for only one period of time. Users requiring longer time must continue bidding until they receive allocation for adequate number of times. In our last work in mechanism design, we investigated the problem of online mechanism design. In an online mechanism, users include the required time in their bids. Once a user wins her bid, she has guaranteed access to the allocated resources for the time period she requested in her bid. In another research work, we investigated the bidding behavior of users where we devised an efficient bidding algorithm for users who would participate in combinatorial auctions in clouds. We believe our work has laid a solid foundation for auction-based allocation in clouds and has opened a number of directions for future research.

1.1 Problem Definition

In this section, we present the part of the problem definition and the notations that are common throughout the dissertation. Specific definitions, assumptions, and notations are covered along with respective research problems that we present.

We assume that the cloud provider allocates its computing resources as m different types of VM instances $VM_1, \ldots, VM_m$. A weight vector $\mathbf{w} = (w_1, \ldots, w_m)$ represents the relative computing power of each type of VM instance. For example, suppose a cloud provider defines three types of VM instances (small, medium and large), which we call VM_1, VM_2 and VM_3. Let the configuration of these VMs be $VM_1 \equiv$ (One 2 GHz processor, 2 GB memory, 1 TB disc storage), $VM_2 \equiv$ (Two 2 GHz processors, 4 GB memory, 2 TB disc storage), and $VM_3 \equiv$ (Four 2 GHz processors, 8 GB memory, 4 TB disc storage). In this case, the weight vector will be $\mathbf{w} = (1, 2, 4)$. Without loss of generality, we assume that $w_1 \leq \ldots \leq w_m$ and $w_1 = 1$. We also assume that there are n users $u_1, \ldots, u_n$ who request bundles of VM instances to the cloud provider by submitting their bids. The bid of user u_j is denoted by $B_j = (r_1^j, \ldots, r_m^j, v_j)$, where r_i^j is the number of VM_i instances user u_j requires in her bundle. v_j is user u_j's valuation for her requested bundle i.e., the maximum amount she wants to pay for it. We define the rest of the terms in the following chapters where we solve different variants of the VM provisioning and allocation problem in clouds.

The first problem we solve as part of our dissertation research is called the Virtual Machine Allocation Problem (VMAP), which we present in Chapter 3. In this problem, we assume that the cloud provider has a fixed number of statically provisioned VM instances to allocate to its users. $k_1, \ldots, k_m$ denote the number of $VM_1, \ldots, VM_m$ instances available for allocation. The objective of VMAP is to determine a set of winners among the users and determine their payment so that the social welfare is maximized, satisfying the constraint given by the available number of VM instances.

The problem we solve in Chapter 4 is to devise an efficient bidding strategy algorithm for users who submit malleable parallel jobs to a cloud computing platform that uses combinatorial auction-based mechanisms to allocate VM instances to its users. Malleable parallel jobs can be executed on any number of processors, given that the parallel speedup decreases non-linearly with increase in number of processors. The objective of the problem is to generate an efficient bid B_j for user u_j given the total workload and degree of parallelism of the job she wants to execute, how much she values the speedup of the execution and her

budget, and the system parameters such as communication overhead.

In Chapter 5, we define the Dynamic VM Provisioning and Allocation Problem (DVMPA) where the number of different types of VM instances are provisioned dynamically based on the market demand. In DVMPA, the total amount of computing resources is denoted by M, where M is the total number of VM_1 instances that can be provisioned using the entire resources available to the cloud provider. Note that we assume that VM_1 is the least powerful VM instance with weight $w_1 = 1$. In DVMPA, we also consider the cost of operating the VM instances: c_I is the cost for an idle VM_1 instance for a 'time unit' and c_R is the cost for a 'running' (i.e., allocated to a user) VM_1 instance for a 'time unit'. Cost of other types of VM instances are linearly proportional to their weight. Given the above parameters, the objective of DVMPA is to determine a set of winners from the users, determine their payments, and determine the number of each type of VM instances to provision so that the the profit (revenue – cost) of the cloud provider is maximized.

We define and solve the problem called Online VM Provisioning and Allocation Problem (OVMPA) in Chapter 6. Unlike the above problems, OVMPA considers the duration of time for which users request the VM bundle. Each user u_j submits her bid to the mechanism where she also includes the number of time units she requires to complete her application l_j and the deadline for task completion d_j. The time instance a_j the user submits her bid is her 'arrival time'. The user's bid and her arrival time constitute her type. An online mechanism is invoked as soon as there are some outstanding bids and there are available resources in the cloud. Therefore, computation of the mechanism is performed on incomplete information i.e., before collecting information on all users and having all resources available. The performance of an online mechanism is worse than offline mechanisms in terms of revenue, but it provides more user satisfaction by guaranteed time and lower average payments. We design the mechanism MOVMPA to solve the OVMPA problem in Chapter 6 and discuss its properties.

1.2 Outline of the Dissertation

In this section, we outline our dissertation and summarize our contributions. In the following we summarize the four research projects that we accomplished as part of this dissertation.

- **Combinatorial Auction-Based Allocation of Virtual Machine Instances in Clouds.** We studied the fixed-price based VM instance allocation mechanisms in clouds and modeled the VM instance allocation problem. We assumed that the cloud providers have fixed number of different types of VM instances to allocate to their users. The users bid periodically to get bundles of VM instances required for their tasks. The cloud providers' goal is to allocate the VM instances to the users who value them the most, determine their prices, and maximize the revenue in the process. We designed two combinatorial auction-based mechanisms CA-LP and CA-GREEDY to solve the problem. We analyzed the mechanisms and proved that they are incentive compatible. Finally, we ran extensive simulation experiments to compare CA-LP and CA-GREEDY with the fixed-price based mechanisms currently used in clouds. Our results showed that CA-LP performs best in all metrics than CA-GREEDY and fixed-price based mechanisms, but CA-GREEDY achieves comparable performance with a much smaller time complexity. We conclude that CA-GREEDY should be used as a general-purpose mechanism for allocating VM instances in clouds. We present this research in Chapter 3. A paper describing this research was published in the Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom 2010) [71] and also received the Best Student Paper Award. An extended version of this paper is under review for the Journal of Parallel and Distributed Computing.

- **Efficient Bidding for Virtual Machine Instances in Clouds.** In order to successfully implement VM provisioning and allocation in clouds, it is crucial that the users can easily participate in such mechanisms and get the best value out of them.

In this research work, we show how a user can efficiently bid for the bundle of VM instances to submit their jobs to the cloud computing platform. We considered a setting where a cloud provider implements the CA-GREEDY mechanism and users submit their bids to secure resources to run their malleable parallel jobs on the cloud. Malleable parallel jobs can be run on any number of processors, with a non-linear relation of the number of processors used to the speedup of job execution. We designed an efficient bidding strategy algorithm called EBS that takes into account the job characteristics, system parameters, and user preferences to generate a bid that comprises the best bundle for a particular job and the true valuation for that bundle. This bid is submitted to the auction to maximize the user's utility. We proved that EBS has reasonable time complexity and experimentally showed that it outperforms a naive bidding strategy. The results of this research were published in Proceedings of the 4th IEEE International Conference on Cloud Computing (IEEE CLOUD 2011) [73]. Later, we extended this paper with more experiments investigating the results with varying the simulation parameters in more dimensions. We submitted the extended version to Journal of Parallel and Distributed Computing and it is currently under review. We present this work in detail in Chapter 4.

- **Combinatorial Auction-Based Dynamic VM Provisioning and Allocation in Clouds.** In the previous research, we considered that the cloud providers statically provision the VM instances and the mechanism allocates them to the users. In this research, we incorporated the dynamic provisioning into the combinatorial auction-based mechanism. We formulated the problem such that the entire resources are considered as 'liquid' resources and the number of different VM instances to provision are determined based on the user demand. We also considered the costs for a VM instance while it is kept idle and when it is allocated to a user. These costs are used to determine a 'reserve price' to avoid revenue losses. We designed a mechanism called CA-PROVISION that performs a combinatorial auction that considers the dynamic provi-

sioning and the reserve price. CA-PROVISION computes the number of VM instances to provision and allocate to the users and the payment to be charged to each user in order to maximize the profit of the cloud provider. We compare the performance of CA-PROVISION with that of CA-GREEDY by performing extensive simulation experiments using traces of real workloads from the Parallel Workloads Archive [23]. The results show that CA-PROVISION significantly improves the resource utilization and percentage of served users when compared to CA-GREEDY. The profit obtained by CA-PROVISION was higher in cases with high density of user demands. And CA-PROVISION obtains higher profit in half of the cases where the user demands are low. We conclude that CA-PROVISION performs better in general, where CA-GREEDY performs better where the pattern of the user demands match the available resources. We present this work in detail in Chapter 5. A paper describing this research was published in the Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (IEEE CloudCom 2011) [72]. An extended version of this paper is under review for IEEE Transactions on Parallel and Distributed Systems.

- **Online Mechanism for VM Provisioning and Allocation in Clouds.** An online mechanism computes the allocation and payment of items without having complete information. In the context of cloud virtual machine allocation, an online mechanism would be invoked as soon as some user submits her bid or some resources become available for allocation. In a cloud computing platform, an online allocation mechanism would have positive effects such as shorter waiting time for the users, reduced idle time of computing resources, etc. However, since an online mechanism works with incomplete information (offline mechanisms are invoked periodically at fixed intervals, therefore they have complete information about the bids that arrived during the past interval), it cannot achieve the same efficiency as the offline mechanisms. To ensure performance, an online mechanism must have a good competitive ratio. In this research project, we formulated the problem of online VM provisioning

and allocation in clouds. In this model, the users include the required time and deadline for their tasks in their bids. We designed the online mechanism called MOVMPA to address this problem. MOVMPA is invoked whenever there are outstanding bids and available resources in the system. It allocates the VM instances to the winning users for the entire period they request them for. It also calculates the payment based on the critical value payment in the online setting. We proved that MOVMPA has good competitive ratio, reasonable runtime, and is truthful. We evaluated this mechanism with extensive simulation experiments using real workload data. We compared MOVMPA with CA-PROVISION and found that MOVMPA performs better than the theoretical competitive ratio, while providing better user experience. Our work was published in the Proceedings of the 5th IEEE International Conference on Cloud Computing (IEEE CLOUD 2012) [74]. We also prepared an extended version of this paper for submission to the IEEE Transactions on Parallel and Distributed Systems. We present this research in Chapter 6.

1.2.1 Our Contribution

In this dissertation, we addressed an emerging problem in cloud computing using game theory and mechanism design techniques. We identified that cloud providers would require combinatorial auction-based mechanisms to determine efficient allocation of resources to their users and to increase their revenue in the process. Combinatorial auctions enable users to express their demand of resources and their willingness to pay in a meaningful way. On the other hand, auction-based provisioning and allocation mechanisms enable the cloud providers determine their prices of resources dynamically and also provision to them according to market demand. In the rapidly improving computing industry, where the price and utility of a computing resources changes continuously, it is a difficult task for a cloud provider to accurately predict demand and price their resources accordingly. Combinatorial auction-based mechanisms allow creating a market that takes care of these complex tasks automatically. In our dissertation, we investigated the problem in different settings. We

designed two combinatorial auction-based mechanisms that allocate pre-provisioned VM instances to users requesting bundles of VM instances. Then, we focused our attention on systems where VM instances are dynamically provisioned based on the current market demand. Finally, we developed an online mechanism for dynamic provisioning and allocation of VM instances. We performed theoretical analysis and simulation experiments with all the above mechanisms. These results will enable a system designer to implement or adapt one of the mechanisms based on their specific needs. We also investigated this problem from the users' point of view. We designed an efficient bidding strategy algorithm for submitting malleable parallel jobs in clouds. We also provide theoretical and experimental results pertaining to this algorithm. We believe our research will encourage implementation of auction-based mechanisms in clouds and also initiate other research in this field.

1.3 Organization

The rest of the dissertation is organized as follows. In the next chapter, we present background knowledge on combinatorial auctions, which is the foundation of this dissertation. We also present a discussion of the related work in the existing literature in Chapter 2. In Chapter 3, we present our research on designing combinatorial auction-based mechanisms where the VM instances are pre-provisioned. We design two mechanisms CA-LP and CA-GREEDY, prove their theoretical properties, and present simulation results comparing them with fixed-price mechanisms. In Chapter 4, we present our research on designing an efficient bidding algorithm to request VM bundles for malleable parallel jobs. We show experimentally that our algorithm, called Efficient Bidding Strategy (EBS), outperforms a naive strategy in terms of generating higher utility for the users. In Chapter 5, we present the design of a dynamic provisioning and allocation mechanism called CA-PROVISION. We present an online mechanism for dynamic provisioning and allocation in clouds in Chapter 6. In Chapter 7 we describe the possible future directions of our research. We conclude the dissertation in Chapter 8.

CHAPTER 2: BACKGROUND

In this section, we briefly introduce the background on combinatorial auctions, which is the foundation of this dissertation. We discuss further details, when necessary, along with the specific research work we present. We also present the literature survey related to our work in this chapter.

2.1 Combinatorial Auctions

Auctions are economic mechanisms that allocate an item or a set of items to participating users. Users express their requests in terms of 'bids', which consist of a valuation (i.e., how much she values an item or what is the maximum amount she wants to pay for it) and a subset of the items (where a set of items are being auctioned). Combinatorial auctions allocate a set of items to users, where users express their bids in terms of bundles of items and their valuation of respective bundles. A user is called 'single-minded' if she requests only a single bundle and is not interested in any subset of it. Formally, a combinatorial auction allocates a set of items S to n users $j = 1, \ldots, n$. We assume that the users are single-minded and each user j submits a bid $\hat{\theta}_j = (\hat{S}_j, \hat{v}_j)$ to the mechanism. Here, $\hat{S}_j \subseteq S$ is the set of items user j requires and $\hat{v}_j$ is the maximum amount she is willing to pay for it. The goal of the mechanism is to determine a set of winners $W \subseteq \{1, \ldots, n\}$ and their payments $p_1, \ldots, p_n$. The users in the set of winners receive their requested bundles and pay the price determined by the mechanism. In most mechanisms, the losing users do not pay anything.

The 'type' or the 'true bid' of a user j is $\theta_j = (S_j, v_j)$, which she may or may not report truthfully to the mechanism. Components of θ_j means that user j truly requires bundle $S_j \subseteq S$ and she receives a value of v_j if she wins her bid. We express this as the valuation

function shown below

$$V_j = \begin{cases} v_j & \text{if } j \in W \\ 0 & \text{otherwise} \end{cases}$$

The quasi-linear utility of user j is the difference between her valuation function and the payment determined by the mechanism, which is expressed as

$$U_j = V_j - p_j$$

We assume that the users are *rational*, i.e., they may bid untruthfully if it benefits them to do so. The goal of the mechanism is to achieve a specific system-wide goal with the reported values and provide incentives so that the users obtain the maximum benefit only by reporting their true values. We elaborate on this in the following section.

2.2 Goals, Properties and Challenges of Mechanism Design

The most reasonable objective function would be to maximize the cloud provider's revenue, but very little is known about revenue maximization in the context of combinatorial auctions [44]. Therefore, the most common goal of combinatorial auction mechanisms is to maximize the so-called *social welfare*, which is the sum of the reported valuations of the winning bidders. This has a positive influence on the revenue, because valuations are the maximum amounts users are willing to pay and they play an important role in determining the payment. Therefore, assuming that the mechanism allocates non-conflicting bundles to users, the goal of a combinatorial auction-based mechanism is to solve the following optimization problem.

$$\max \sum_{j \in W} \hat{v}_j \tag{2.1}$$

subject to,

$$\bigcup_{j \in W} \hat{S}_j \subseteq S \tag{2.2}$$

$$\hat{S}_i \cap \hat{S}_j = \emptyset \qquad \text{where } i \neq j \tag{2.3}$$

Equation (2.1) states that the objective of the mechanism is to maximize the social welfare. The constraint shown in Equation (2.2) is that the total resources allocated must adhere to the total resources available. Equation (2.3) ensures that no item is allocated to multiple bidders.

To achieve the above goal, the mechanism must provide incentives to the users so that they report their values truthfully. A mechanism is *truthful* if a user can maximize her utility by reporting her true value to the mechanism, irrespective of the bids of the other users. Another desirable property of a mechanism is that it should be *individually rational*, which means a user truthfully reporting her valuation will never incur a loss by participating in the mechanism, irrespective of the other users' bids.

There are certain challenges that we need to overcome while designing a combinatorial auction-based mechanism. First of all, combinatorial auctions are NP-hard problems, therefore requiring us to find an efficient approximation algorithm for allocation. On the other hand, classic auction-design techniques to achieve truthfulness do not directly apply to approximate solutions. Archer and Tardos [8] showed that to design an approximation mechanism, the allocation algorithm must be *monotone* and the payment must be determined using the *critical value* method. We discuss monotonicity and critical value in the context of the specific research problems later.

2.3 Related Work

Auction-based allocation of computing resources has been widely studied in the literature, especially in the distributed computing setting. One of the earliest use of auctions in

computing was in reserving computing time of a shared minicomputer at Harvard University [60]. In this auction, an artificial 'bidding currency' was provided to the users in order to participate in the auction to reserve computing time for them. Gagliano [26] also investigated the allocation of computing resources through auctions, where the tasks themselves are provided enough intelligence to calculate the bid that is necessary to get the required resources.

Auctions have been widely studied for scheduling and resource allocation in computational grids. Wolski *et al.* [68] compared commodities markets and auctions in grids in terms of price stability and market equilibrium. Gomoluch and Schroeder [29] simulated a double auction protocol for resource allocation in grids and showed that it outperforms the conventional round-robin approach. Garg *et al.* [27] designed a double auction-based meta-scheduler for grids, which schedules grid jobs into different clusters that improves both user utility and system performance when compared to traditional meta-schedulers. Das and Grosu [18] proposed a combinatorial auction-based protocol for resource allocation in grids. They considered a model where different grid providers can provide different types of computing resources. An 'external auctioneer' collects this information about the resources and runs a combinatorial auction-based allocation mechanism where users participate by requesting bundles of resources. The major difference between our research in combinatorial auctions and the one presented in [18] is that we are considering allocating VM instances of a single cloud provider whereas Das and Grosu [18] considered the problem of allocating different types of physical resources from multiple grid providers. Dash *et al.* [19] formulated a mechanism design problem for task allocation in grids. They considered an optimization problem where the goal is to reduce the overall system cost, but selfish resource providers may misreport their capacity and cost parameters if they benefit by doing so. They proposed a centralized and a distributed mechanism to solve this problem. A system architecture for incentive-compatible resource allocation in grids was proposed by Grosu [30]. The proposed architecture allows both users and providers to deploy and participate in different mechanisms that determine resource allocation and

pricing.

Recently, researchers investigated the economic aspects of cloud computing from different points of view. Wang *et al.* [66] studied different economic and system implications of pricing resources in clouds. They performed experiments on Amazon EC2 and on their own testbed concluding that the pricing scheme used by Amazon is unfair to the users. Walker *et al.* [64] proposed a model to determine the benefits of acquiring storage services from clouds. The tool CloudCmp [36] was developed to assist users in choosing the appropriate service providers based on the user's requirements.

Buyya *et al.* [10] proposed an infrastructure for auction-based resource allocation across multiple clouds. Altmann *et al.* [1] proposed a marketplace for resources where the allocation and pricing are determined using an exchange market of computing resources. In this exchange, the service providers and the users both express their ask and bid prices and matching pairs are granted the allocation. Risch *et al.* [52] proposed a testbed for cloud services designed for testing different mechanisms. They deployed the exchange mechanism proposed by Altmann *et al.* [1] on this platform. A marketplace proposed in the above research work is not easy to achieve because of interoperability issues of current cloud platforms. The current focus on the cloud markets is mostly on the single provider and multiple users model. Our research is therefore based on the single-provider multiple-users model. A combinatorial exchange would be a possible extension of our work towards the federated cloud platforms.

There have been significant efforts in designing auction-based allocation mechanisms for clouds. Lin *et al.* [37] proposed the use of a simple $(k+1)^{th}$ price auction for allocating cloud resources. They showed by statistical analysis that when there is a large number of resources and users, the auction can obtain an efficient allocation and a reasonable revenue for the cloud provider. To the best of our knowledge, Amazon EC2 implemented the first auction in clouds, named Spot Instances. Unfortunately, the mechanism behind the Spot Instances is not publicly available. Researchers reported work on investigating the Spot Instances and using them efficiently. Chohan *et al.* [16] showed how to accelerate

MapReduce jobs using Spot Instances. They also analyzed the performance gain and the cost effectiveness of this approach. Ben-Yehuda *et al.* [9] analyzed the pricing of Amazon EC2 and claimed that it is not market-driven. They showed that the prices are randomly generated considering a hidden reserve price that is not driven by supply and demand.

Campos-Náñez *et al.* [11] investigated dynamic auction settings for 'utility computing', where the bidders are the service providers (service queues) that bid for one customer's job at a time. In their model, the available capacity of the queues is public knowledge and the bidders only bid their prices. The service with the lowest bid is selected for the customer that arrived. The authors showed that there exists a Markov Perfect Equilibrium for the game.

The differences between the marked-based mechanisms designed for grids and those designed for clouds are mainly related to their underlying resource allocation model. Clouds allocate resources in terms of VM instances while traditional grids allocate physical resources directly without involving virtualization. The market-based mechanisms are more suitable for clouds since they are designed to make profit by selling services while traditional grids were designed mainly for sharing resources and not for making profits by selling resources.

One of our research work deals with dynamic provisioning of VM instances, hence we discuss some existing literature on VM provisioning here. Researchers approached the problem of VM provisioning in clouds from different points of view. Shivam *et al.* [58] presented two systems called Shirako and NIMO that complement each other to obtain on-demand provisioning of VMs for database applications. Shirako does the actual provisioning and NIMO guides it through active learning models. The CA-PROVISION mechanism we propose here performs both demand tracking and provisioning via a combinatorial auction. Dornemann *et al.* [21] proposed on-demand resource provisioning for the Business Process Execution Language (BPEL). Their work extends BPEL engine so that it can support scientific workflows by dynamically provisioning resources from Amazon EC2 when the demand surpasses the capacity of the BPEL host.

Dynamic provisioning of computing resources was investigated by Quiroz *et al.* [50] who proposed a decentralized online clustering algorithm for VM provisioning based on the workload characteristics. The authors proposed a model-based approach to generate workload estimates on a long-term basis. Our proposed mechanism provisions the VMs dynamically and it does not require the prediction of the workload characteristics, rather the current demand for VMs is captured and the provisioning is decided by a combinatorial auction-based mechanism. Van *et al.* [62] proposed an autonomic resource management system that decouples VM allocation from the physical mapping of instances to resources. They showed that their approach can simultaneously satisfy both service level agreement and resource utilization criteria. Vecchiola *et al.* [63] proposed a deadline-driven provisioning mechanism supporting the execution of scientific applications in clouds.

The goal of our research on auction-based mechanisms is to improve the efficiency of cloud resources and the revenue and/or profit of cloud providers. Other researchers investigated this area using different approaches. For example, some research work investigated methods to increase the efficiency of the cloud data centers. Kansal *et al.* [33] extended the power metering technique of physical computing resources towards a per-VM power metering system. They proposed a metering capability for VM power capping which reduces the power provisioning costs in data centers. Meng *et al.* [38] proposed a technique that finds and exploits complementary patterns of workloads to multiplex virtual machines. This technique 'packs' multiple VMs into a smaller set of resources while maintaining the quality of service.

Chen *et al.* [15] combined several factors of cost saving and optimal resource utilization to minimize the cost for cloud providers and maximize their profit. Their solution combines the use of vector arithmetics to ensure balanced utilization of computing resources and efficient VM reconfigurations at runtime. Ghosh and Naik [28] utilized the fact that most users request more resources than their application actually require to devise a strategy to 'over-commit' cloud resources in order to maximize profit. Lampe *et al.* [34] presented an equilibrium auction for allocating VM instances in cloud and showed that a heuristic

algorithm performs much faster with little degradation in performance. But their approach considers a bid to be a collection of many individual bids that request only one VM instance. But, unlike combinatorial auctions, this equilibrium price auction cannot guarantee that a user requesting a bundle will receive the entire bundle even if her bid satisfies the equilibrium price. However, the authors did a good job in considering the capacity of physical machines to determine the number of VM machines the cloud provider can allocate. Tsai and Qi [61] developed a pricing strategy for cloud services to ensure fair pricing in a dynamic setting. Zafer *et al.* [70] looked at minimizing the cost of users by using statistical analysis to design a cost-effective bidding strategy for Amazon Spot Instances. Menychtas *et al.* [39] proposed a framework for a cloud marketplace that will enable trading of cloud services among multiple providers and users, while incorporating business terms (e.g., SLAs) into the trading model.

The complexity of solving the combinatorial auctions, specifically the winner determination problem, was first addressed by Rothkopf *et al.* [53]. Sandholm [55] proved that solving the winner determination problem is computationally hard. Rothkopf *et al.* [53] and Sandholm [55] used the technique of pruning the search tree to devise approximation algorithms. Andersson *et al.* [6] proposed an integer programming based solution to the winner determination problem.

Zurel and Nisan [75] also presented an efficient algorithm for combinatorial auctions. Lehmann *et al.* [35] studied combinatorial auctions with single-minded bidders and devised a greedy mechanism for combinatorial auctions. We extend this mechanism in Chapter 3 to design the CA-GREEDY mechanism. Archer *et al.* [7] considered another case of single-minded bidders where multiple identical copies are available for different types of items. They provided a mixed integer programming based algorithm for winner determination and showed theoretically that their solution performs better than generalized solutions for this special case. We extend this mechanism such that it can be used to solve the VM allocation and pricing problem we consider in Chapter 3. A detailed survey on combinatorial auctions can be found in [20]. Cramton *et al.* [17] provides good foundational knowledge on this

topic.

Recently, several researchers investigated the design of online mechanisms in different contexts. Parkes and Singh [48] designed a Markov Decision Process-based online mechanism and later on provided an approximate solution for the same model for large problems [49]. Hajiaghay *et al.* [31] proposed the idea of automated online mechanism design. Fundamentals of online mechanism design are covered in [47]. Carroll and Grosu [12] designed an online mechanism for scheduling malleable parallel jobs on parallel systems. They considered preemption of jobs in their model in order to provide for jobs with higher valuation that are submitted later than the currently allocated jobs.

The marketing and advertising in the Internet has created a huge market driven by ad auctions [22]. Chen *et al.* [14] investigated Internet auctions in a double-auction setting where multiple copies of one item are sold. They showed that bidding the actual value of the item is a weakly dominant strategy for the bidders in this setting. The development of cloud computing initiated research in developing and analyzing the market for computing resources and services. Wang *et al.* [65] devised a 'requirement-based' bidding language for combinatorial auction-based scheduling problems. The proposed bidding language enables the bidders to express otherwise complex requirements (e.g., specifying the required makespan for the set of jobs submitted) as an atomic bid. They also developed a branch-and-bound algorithm to solve the winner determination problem for this setting and reported significant performance improvement over a commercial solver.

Recently, many researchers have investigated the design of efficient bidding strategies for combinatorial auctions. Sui and Leung [59] proposed an adaptive bidding strategy for combinatorial auctions. They considered a multi-round first-price combinatorial auction for allocating computing resources. In their proposed bidding strategy the bidder updates her bid dynamically based on the outcome of the previous bidding round, in order to maximize her utility. A similar idea is used by Yi *et al.* [69] to design a strategy for bidding in the Amazon EC2 Spot Instances auction [3]. The authors use statistical analysis to show how a task can be completed at reduced cost and without premature termination on the Amazon

EC2 cloud. Our work differs from the above two in that we are considering a combinatorial auction mechanism which guarantees the maximum utility for a user when she bids truthfully. Therefore, our proposed bidding strategy is based on determining the user's 'true' valuation for a bundle and it does not require knowledge of past auction outcomes. This is a big advantage since the users do not require information from past auctions when making bidding decisions. An *et al.* [5] proposed a bidding strategy algorithm that considers a given 'synergy' value between different items, when calculating the valuation of a bundle of items in general combinatorial auctions. In our work in Chapter 4, we defined the synergy between the VM instances in a bundle based on a speedup function that takes into account the overheads of execution on multiple VM instances. The speedup function we use is an extension of the speedup function for malleable parallel jobs proposed by Havill and Mao [32]. Carroll and Grosu [12] considered the problem of scheduling malleable parallel jobs and designed incentive-compatible scheduling mechanisms to solve it. A comprehensive survey classifying parallel applications and their scheduling strategies can be found in [25].

Other related research includes designing an agent for bidding in combinatorial auctions in grids [56] and proposing an efficient policy for obtaining cloud resources for large applications [13]. Developing scheduling algorithms that satisfy the budget constraint of the users were investigated by Oprescu and Kielmann [46] and by Shi and Hong [57]. The bidding strategy that we propose in Chapter 4 also considers the budget constraints, but as opposed to the work reported in [46] and [57], the users do not need to submit their budget information to any scheduler. In our work in Chapter 4, the bidding algorithm will generate the bid so that the budget constraint is satisfied. Therefore, the scheduler is not involved in checking and guaranteeing that the budget constraint is satisfied.

CHAPTER 3: COMBINATORIAL AUCTION-BASED ALLOCATION OF VIRTUAL MACHINE INSTANCES IN CLOUDS

3.1 Introduction

Cloud computing enables individuals and small to medium enterprises satisfy their computational needs with no or minimum upfront costs of acquiring hardware and software. On the other hand, cloud providers benefit by commercializing their huge computing resources through the cloud computing platform. A cloud computing platform abstracts the underlying physical resources from the users by providing them with the view of virtual machines (VMs). This enables easy management and pricing of the resources. Currently, the majority of cloud providers price their computing resources based on the 'size' of the VM instances offered. They define different types of VM instances by specifying the number and speed of processors, the memory size, the bandwidth allocation, etc. There are two ways to 'purchase' the VM instances: pay as you go and long term contract. In both cases users pay fixed prices per unit of time for using the resources; the only difference is that by committing to a long term contract they usually pay less per unit of time for using the same resource.

We argue that the currently used fixed-price schemes for allocation and pricing of resources have several drawbacks. First, they are not economically efficient [67], that is, they cannot guarantee that the user who values a bundle of VM instances the most gets it. Second, fixed prices do not necessarily reflect the equilibrium prices that arise from market

demand and supply. This may lead to lower than optimal revenue for the service providers. Finally, since in cloud computing platforms resources are sold for a period of time, it is desirable that user requests be evenly distributed throughout the day. In general, the current fixed-price methods do not provide users with incentives for demand shaping (i.e., selecting their execution time-frames in such a way that the system load is balanced over time). It is possible to modify a fixed-price mechanism so that it provides such incentives by setting up different fixed prices at different times of the day based on historical demand. This needs statistical analysis and adjustment of prices as the demand pattern changes making it hard to achieve dynamic price adjustment.

The inefficiencies in solving the resource allocation problem in clouds mentioned above can be best addressed by employing auction-based mechanisms. Among different types of auctions, the *combinatorial auctions* are the most suitable for solving the VM pricing and allocation problem in clouds. In combinatorial auctions, the participants bid for bundles of items rather than individual items [17]. This enables bidders to express their valuations in a more meaningful way, especially when the items they require are complementary to each other. To illustrate this, let us consider the following example. A cloud service provider offers 'small' and 'large' VM instances. Suppose a user wants to deploy a three-tier web application on the cloud. The application needs a database server, an application server, and two web servers. The database and application servers are heavy weight and therefore the user prefers large instances for them. The web servers are light weight and can be hosted on two small instances. Thus, a user needs to run an application which requires two small and two large VM instances. It is more meaningful for her to be able to bid for the entire bundle she needs rather than bidding for each VM instance separately. Bidding for each VM instance separately involves the risk of ending up acquiring just a subset of her required set of VM instances. The motivation behind our work is that by designing and deploying combinatorial auction-based mechanisms for allocating VM instances, the cloud providers can guarantee fairness to their users as well as enjoy higher revenues and a balanced load on their systems over time. Load balancing over time is actually a side-effect

of using auctions for allocating VM instances. Users with lower valuations for the VM instances will choose a time-frame that does not conflict with that of 'high valuation users'. For example, if large businesses request resources during the daytime, individual users may consider that the nighttime slots are more suitable for them, thus balancing the load of the system over time.

Application of auctions, however, is not entirely new to the cloud computing community. After allocating computing resources for the long-term and on demand users, Amazon EC2 sells the remaining virtual machines (instances) through an auction called Spot Instances [3]. In this auction, the bidders specify their demand (i.e., the number and the type of instances) and the maximum price they are willing to pay. Amazon periodically runs the auction with active bidders to determine the current price and then users with bids higher than that price are provided with their desired instances. All users pay the same price per instance which is computed by the auction. A user getting the allocation may be terminated at a later point if the auction-determined price goes beyond her bid. This approach is different from combinatorial auctions because one single price is determined based on market supply and demand (i.e., equilibrium) and all bidders pay the same price per item regardless of how much they value the item. On the other hand, in combinatorial auctions, each winning bidder's payment is calculated based on her and other bidders' valuations. Another important difference is that the Spot Instances auction does not support bidding on bundle of instances, while combinatorial auctions were specifically designed to work with such bundles. From Amazon's initial effort of using auction-based allocation, it is reasonable to expect that cloud providers will be interested in more efficient allocation and pricing schemes in the near future. Combinatorial auctions will clearly be one of the most desirable allocation schemes in this regard. This is supported by their successful application in various fields ranging from selling wireless spectrum to transportation procurement for large industries [17].

3.1.1 Our Contribution

We formulate the problem of allocating VM instances in clouds as a combinatorial auction problem. The objective of this problem is to efficiently allocate VM instances of several types to several users requesting a set of VM instances of different types. To solve this problem, we propose two combinatorial auction-based allocation mechanisms. These two mechanisms are obtained by extending the mechanisms proposed by Archer *et al.* [7] and Lehmann *et al.* [35]. The mechanism proposed by Archer *et al.* [7] considers a combinatorial auction problem where a user can include at most one item of a particular type in her requested bundle. We relax this condition to allow users requesting more than one item of a given type. Also, the mechanism proposed by Archer *et al.* [7] is suitable for combinatorial auctions with many types of items where each type of items has few instances. We extend the mechanism so that it can be applied to the VM allocation problem where there are few types of items and many instances of each type.

The other mechanism we propose is an extension of the greedy mechanism proposed by Lehmann *et al.* [35]. This mechanism determines the allocation based on the valuation of the users and the total number of items they request. We extend the mechanism proposed by Lehmann *et al.* [35] so that it considers the relative sizes of the VM instances and show that the properties of the original mechanism are maintained. We compare the two proposed combinatorial auction-based mechanisms with the fixed-price based allocation mechanism used by Microsoft in their Windows Azure platform [41]. We investigate the relative performance of these three allocation mechanisms by performing extensive simulation experiments. We also consider variants of the fixed-price mechanism in which the fixed prices are different at different times of the day. We compare the performance of these mechanisms with that obtained by our proposed mechanisms as well. The experiments show that the proposed combinatorial auction-based mechanisms clearly outperform the fixed-price mechanism in terms of resource utilization, generated revenue, and allocation efficiency. We analyze the results and provide recommendations on where to use the proposed mechanisms.

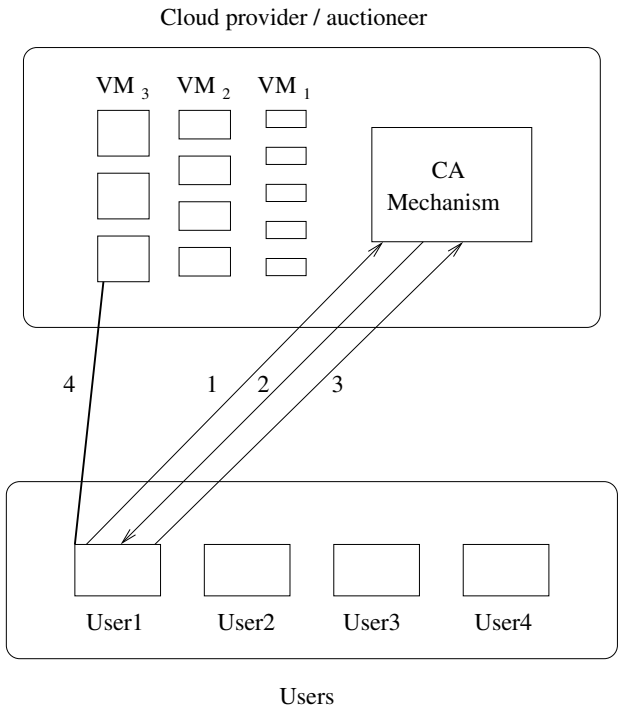


Figure 3.1: VM instance allocation in clouds: system model

3.1.2 Organization

The rest of the chapter is organized as follows. In Section 3.2, we formally define the VM instance allocation problem. In Section 3.3, we present the mechanisms we consider for solving the VM allocation problem. In Section 3.4, we describe the experimental results. We conclude this chapter by summarizing the contributions in Section 3.5.

3.2 Virtual Machine Allocation Problem

The cloud providers set different configurations of VM instances that the users can request. A user requests VM instances of different types and pays the cloud provider for the time she uses them. Usually, the prices for different types of instances for short-term use are fixed

by the cloud providers in advance. Another possibility is that a user sets up a long-term contract if she requires the resources for a long period of time, in which case she may obtain them for a lower price. Here we consider the problem of efficient allocation and pricing of VM instances for short-term use.

In Figure 3.1, we provide a high-level representation of the VM instance allocation system we consider. The cloud provider has several VM instances of different types available for allocation and runs a combinatorial auction-based mechanism to allocate them to users. The auction mechanism consists of three steps. First, the mechanism collects 'bids' from the users, which comprise the number of different types of VM instances a user requests and the price she offers for that bundle. Then, the mechanism computes the allocation and the payment based on the collected bids and the availability of resources. Finally, users who get the allocation pay the cloud provider and obtain access to the resources they requested.

We define the *Virtual Machine Allocation Problem (VMAP)* as follows. Assume that the allocation and prices are decided periodically by a given mechanism. Let the interval between two such decisions be 'one unit of time'. VMAP considers allocating the VM instances for one unit of time. Assume that a cloud provider has m different types of virtual machines $VM_1, \ldots, VM_m$. The relative computing capabilities (based on number and speed of CPUs, memory, etc.) of these VMs are characterized by a vector $\mathbf{w} = (w_1, \ldots, w_m)$, where $w_i \in \mathbb{R}_+$, $i = 1, \ldots, m$. We also assume that $w_1 = 1$ and $w_1 \leq w_2 \leq \ldots \leq w_m$. To illustrate this, we consider the types of instances currently offered by Microsoft Azure Platform: Small (CPU 1.6 GHz, Memory: 1.75 GB, Storage: 225 GB), Medium (CPU 2x1.6 GHz, Memory: 3.5 GB, Storage: 490 GB), Large (CPU 4x1.6 GHz, Memory: 7 GB, Storage: 1 TB), and Extra large (CPU 8x1.6 GHz, Memory: 14 GB, Storage: 2 TB). In this example, VM_1, VM_2, VM_3, VM_4 are the Small, Medium, Large, and respectively Extra large VM instances. The weight vector characterizing the VM instances is $\mathbf{w} = (1, 2, 4, 8)$.

Let us assume that k_i copies of VM_i instances are available for allocation at a given instance of time, $i = 1, \ldots, m$. There are n users $u_1, \ldots, u_n$, each requesting a set (bundle) of VM instances and revealing how much she values that particular set. That is, a user u_j

is requesting VMs from the cloud provider by placing a bid $B_j = (r_1^j, r_2^j, \ldots, r_m^j, v_j)$, where $r_i^j \in \{0, 1, \ldots, k_i\}$ is the number of instances of type VM_i user u_j requires in her bundle and v_j is her valuation for this bundle, i.e., the maximum price she is willing to pay for using the requested VMs for one unit of time. Here we consider the users to be single-minded bidders. A single-minded bidder u_j desires only a specific bundle of items S_j, and values that bundle at v_j. Thus, u_j has the following valuation function for a bundle S [35],

$$v(S) = \begin{cases} v_j & \text{if } S_j \subseteq S \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

We would like to mention that the assumption of single-minded bidders does not limit the users to express more flexible requirements. Our model assumes that auctions are run periodically and that bidders will request only one bundle in a given auction. Since the auctions are run periodically, a user may choose to revise her bid based on the previous auction outcome and her preference. For example, suppose that the time interval between consecutive auctions is one hour. If a user needs a particular bundle for five units of time and her deadline to complete the job is ten hours, she needs to win five auctions within ten hours. She may choose to bid the same value until her job is finished, or she may choose to start with a low bid and raise it when the deadline is approaching. Users executing parallel applications may want to request as many VM instances as possible to finish their jobs quickly. In this case, they could start by bidding for the largest possible bundle they can afford and if not successful, adjust the requested bundle size for the next auction. If a user must require continuous allocation of resources, she may continue bidding increasing values in order to increase her chances of winning every auction.

The goal of the VM allocation problem, given the set of users U and their bids, is to determine the set of winners $W \subseteq U$ and the price the winners have to pay to the cloud provider. User u_j is a winner (i.e., $u_j \in W$) if she receives her requested bundle of VM instances. The price user u_j pays to the cloud provider is denoted by p_j. We formally define the VM allocation problem as follows:

Virtual Machine Allocation Problem (VMAP)

Determine the set of winners, $W \subseteq U$, and payment p_j for each user u_j, $j = 1, \ldots, n$, such that

$$\sum_{j:u_j \in W} r_i^j \leq k_i \quad i = 1, \ldots, m \tag{3.2}$$

$$0 \leq p_j \leq v_j \quad \text{if } u_j \in W \tag{3.3}$$

$$p_j = 0 \quad \text{if } u_j \notin W \tag{3.4}$$

The constraint in Equation (3.2) ensures that the users are allocated at most k_i instances of VM_i. Equations (3.3) and (3.4) ensure that the winners pay at most their valuations and the losers do not pay at all.

Note that VMAP does not have an objective function. The most reasonable objective function would be to maximize the cloud provider's revenue, but very little is known about revenue maximization in the context of combinatorial auctions [44]. Combinatorial auctions are usually designed to maximize the sum of the bidders' valuations, i.e., $\max \sum_{j=1}^{n} v_j$. Since valuation is a measure of willingness to pay, maximizing the sum of the valuations usually generates more revenue for the resource provider than a fixed-price allocation does. On the other hand, given the prices of each type of VM instance, a fixed-price allocation mechanism does not have an objective function to maximize. Therefore, VMAP is formulated here as a feasibility problem with the constraints that are to be satisfied by all types of solutions. We shall introduce other constraints and/or objective functions when we discuss the proposed mechanisms for solving VMAP.

3.3 Virtual Machine Allocation Mechanisms

In this section, we present three mechanisms that solve VMAP. The first, called FIXED-PRICE, is the fixed-price mechanism currently used by several cloud service providers [2, 40].

The next two mechanisms are the proposed combinatorial auction-based mechanisms, CA-LP (Combinatorial Auction - Linear Programming) and CA-GREEDY (Combinatorial Auction - Greedy). CA-LP is an extended version of the mechanism proposed by Archer *et al.* [7]. The mechanism proposed by Archer *et al.* [7] solves a problem similar to VMAP by using linear programming relaxation and randomized rounding. We extend that mechanism so that it is able to solve VMAP. CA-GREEDY is an extension of the mechanism proposed by Lehmann *et al.* [35]. The mechanism proposed by Lehmann *et al.* [35] provides the best achievable approximate solution[1] for combinatorial auctions with single-minded bidders. However, this is a general purpose mechanism that does not assume any relative importance of the items being allocated. We extend this mechanism by incorporating the weights of different types of VMs as described in Section 3.2. We now describe each mechanism in detail.

3.3.1 FIXED-PRICE Mechanism

The FIXED-PRICE mechanism presented in Algorithm 1 defines a fixed-price vector $\mathbf{f} = \{f_1, \ldots, f_m\}$, where f_i is the price a user has to pay for using one instance of VM_i for one unit of time. The mechanism allocates VM instances to the users in a first-come, first-served basis until the resources are exhausted. It also makes sure that in order to get the requested bundle, the valuation of user u_j is at least F_j, where F_j is the sum of the fixed prices of each VM instance in her bundle (line 10). It also makes sure that the allocation does not exceed the number of available VM instances of each type (line 11). The set of users receiving the requested bundle is denoted by W. A user pays the sum of the fixed-prices of each VM instance in her allocated bundle.

[1]Lehmann *et al.* [35] showed that the approximation ratio achieved by their proposed mechanism cannot be further improved unless NP = ZPP.

Algorithm 1 FIXED-PRICE Mechanism

{*Phase 1: Receive requests from users*}
for $j = 1, \ldots, n$ **do**
    Receive $(r_1^j, \ldots, r_m^j, v_j)$ from user u_j
end for
{*Phase 2: Allocation*}
Sort users according to their time of placing the request, from earliest to latest. (Here we assume $u_1, u_2, \ldots, u_n$ as the order.)
Initialize $W \leftarrow \emptyset$
for $j = 1, \ldots, n$ **do**
    $F_j \leftarrow \sum_{i=1}^{m} r_i^j f_i$
    if $(v_j \geq F_j)$ **and**
        $(r_i^j + \sum_{u_{j'} \in W} r_i^{j'} \leq k_i,\ i = 1, \ldots, m)$ **then**
        $W \leftarrow W \cup \{u_j\}$
    end if
end for
{*Phase 3: Payment*}
if $u_j \in W$ **then**
    User u_j pays, $p_j = F_j$
else
    User u_j pays, $p_j = 0$
end if

3.3.2 Combinatorial Auction-Based Mechanisms

The general combinatorial auction problem can be informally stated as determining the allocation and prices of bundle of items such that the sum of the user's valuations is maximized. In a combinatorial auction, user valuations are expressed on bundles of items rather than on individual items.

A desired property of a combinatorial auction mechanism is *truthfulness*. A mechanism is truthful if the participants benefit the most when they reveal their true valuations to the mechanism. A participant's benefit in a combinatorial auction is expressed by her *utility*, which is defined as the difference between the valuation she receives from the resource allocation and the price she pays to the mechanism. An ideal truthful mechanism determines the optimal allocation that maximizes the sum of the valuations and computes payments such that each participant maximizes her utility only by reporting her true valuation to the

mechanism. A truthful mechanism helps the bidders in that they do not need to compute a complex strategy or assume other users' strategies while making their bids. They just need to bid their true valuations for the bundle since bidding any other value will not improve their utility.

The winner determination problem of combinatorial auctions is an NP-hard problem [55]. Therefore, research has been conducted to find approximate solutions to combinatorial auctions. In order to obtain a truthful approximation mechanism that solve the winner determination problem, few issues need to be addressed [8]. The approximation algorithm needs to be *monotone*. In a monotone allocation algorithm, a bidder can only increase her chance of getting her requested bundle by reporting a higher valuation or by requesting fewer items in her bundle. A monotone allocation algorithm allows finding the so called *critical value* of a winning bidder, which is the minimum she needs to bid in order to get her requested bundle. In a truthful mechanism a winning user has to pay her critical value to the mechanism. For some combinatorial auction problems, randomization is involved in the winner determination and/or the payment calculation algorithm. In that case, the goal of the resulting mechanism is to ensure that the participants maximize their expected utility by bidding their true values. Such mechanisms are *truthful in expectation*. We discuss the useful properties of our proposed mechanisms in the next subsections.

The proposed mechanisms are intended to be run periodically, each time considering the bids placed by the users during that period. It is assumed that users place their bids until they have been allocated their requested resources for enough units of time to execute their job to completion, or it becomes obvious that their job cannot be completed by a deadline. We also assume that the VM instances are statically provisioned, that is, the cloud provider has already provisioned a given number of VM instances of each type and only these instances are available for allocation.

CA-LP Mechanism

Archer *et al.* [7] considered a combinatorial auction problem similar to VMAP. The difference is that in their case bidders can request at most one copy of each item type (i.e., $r_i^j \in \{0, 1\}$), whereas in the VMAP, users can request multiple copies of each type of item (i.e., $r_i^j \in \{0, 1, \ldots, k_i\}$). We modify the winner determination algorithm of the original mechanism such that it is able to solve VMAP. The algorithm for the calculation of payment is kept the same as in [7] because it maintains its properties when applied to VMAP with the modified winner determination algorithm. We present it here for completeness. The CA-LP mechanism is given in Algorithm 2.

CA-LP involves solving the linear program given by equations (3.5-3.7). The objective of the linear program is to find a vector of 'fractional allocations' $\mathbf{x} = \{x_1, \ldots, x_m\}$ that maximizes the sum of the users' valuations (Equation (3.5)). In line 6, the total number of available VM_i instances is reduced to k'_i, which is then used in the constraint in Equation (3.6). This constraint limits the allocation of VM_i instances to k'_i. Using k'_is instead of k_is in this constraint helps reducing the probability of over allocating the VMs during the randomized rounding performed in lines 9 to 15. This constraint is a modification of the constraint used in the mechanism presented in [7] by letting r_i^j take any value rather than only 0 and 1. The next constraint (Equation (3.7)) bounds the fractional allocation values between 0 and 1.

Lines 9 to 15 implement the randomized rounding where user u_j is selected as a winner with probability x_j, if this allocation does not violate any constraint in Equation (3.6). This operation is executed in order of decreasing x_j so that if there is a violation in the constraint, the user assigned a lower x_j is not included in the set of winners W. This step is another modification of the winner determination algorithm presented in [7]. In the original algorithm, users are first included in W with a probability of x_j and if constraint (3.6) is violated for any item, all users requesting that item are excluded from W. This method is suitable for auctions where many different types of items are sold and each type of item

Algorithm 2 CA-LP Mechanism

{*Phase 1: Collect Bids*}
for $j = 1, \ldots, n$ **do**
  Collect bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$ from user u_j
end for
{*Phase 2: Winner Determination*}
Set $k_i' \leftarrow (1 - \epsilon)k_i$, where $0 < \epsilon < 1$, $i = 1, \ldots, m$
Solve the following linear program

$$\max \sum_{j=1}^{n} x_j v_j \tag{3.5}$$

subject to

$$\sum_{j=1}^{n} x_j r_i^j \leq k_i', \quad i = 1, \ldots, m \tag{3.6}$$

$$0 \leq x_j \leq 1, \quad j = 1, \ldots, n \tag{3.7}$$

Initialize $W \leftarrow \emptyset$
for each user u_j, taken in descending order of x_j **do**
  Generate a random number $y_j \in [0, 1]$
  if $(y_j \leq x_j)$ **and**
    $(r_i^j + \sum_{j':u_{j'} \in W} r_i^{j'} \leq k_i, i = 1, \ldots, m)$ **then**
    $W \leftarrow W \cup \{u_j\}$
  end if
end for
{*Phase 3: Payment* (same as in [7])}
for each user $u_j \in W$ **do**
  Perform binary search for v_j' in the range $[0, v_j]$
  (i) Set valuation of u_j as v_j' in Equation (3.5);
  (ii) Solve the LP, let x_j' be the fractional allocation computed for u_j;
  (iii) Until a v_j' is found such that, setting valuation of u_j less than v_j' generates $x_j' < y_j$ and setting the valuation greater than v_j' generates $x_j' > y_j$. This v_j' is the 'critical value'.
  $p_j \leftarrow v_j'$
end for
for each user $u_j \notin W$ **do**
  $p_j \leftarrow 0$
end for

has only a few copies. But in the context of VMAP, this approach will significantly affect the allocation since each type of VM has many copies and there are only a few different types of VMs. For example, let a cloud provider offer four types of virtual machines, 500 instances of each type. Suppose that after rounding, VM_1 becomes over allocated. The mechanism proposed by Archer *et al.* [7] discards all the users that request any instance of VM_1 in her bundle. This results in 500 unsold VM instances. The other VMs requested by those users are also deallocated. This is the reason we cannot use the original winner determination algorithm proposed by Archer *et al.* [7] to solve VMAP.

The payment is calculated in lines 17 to 23. For each winning user u_j, CA-LP computes u_j's critical payment as follows. It performs a binary search in the range $[0, v_j]$, where v_j is the reported valuation of u_j. For each $v'_j \in [0, v_j]$, it solves the linear program given in line 7 until it finds the minimum v'_j that yields $x_j \geq y_j$ (line 18). This v'_j is the critical value for user u_j because reporting a valuation less than v'_j will not allow her to win the bid, and therefore v'_j is what she has to pay. However, a losing user pays zero (lines 21 and 22). Note that the payment computation phase of CA-LP is the same as in the original mechanism.

We now summarize the changes we made to the original mechanism by Archer *et al.* in order to be able to solve VMAP. First, we relaxed the problem formulation to allow users to request multiple VM instances of the same type in their bundles. This is important in order to provide the user with more flexibility of bidding. The other modification is significant in terms of resource utilization. The original mechanism discards all bids that include a conflicting item. This approach is suitable only in the cases where the auction involves many item types where the number of each type of items is very small. The cloud providers usually offer only a few item types (VM instance types), and a large number of items of each type. Keeping the original approach would result in poor utilization of resources, and thus, we modified the allocation function to address this issue and at the same time maintain the truthfulness property.

Archer *et al.* [7] proved that the original mechanism is truthful in expectation. We

claim that CA-LP maintains this property.

Theorem 1. *CA-LP mechanism is truthful in expectation.*

Proof. In order to prove that an approximation mechanism is truthful, we need to prove that its winner determination algorithm is monotone and that the payment calculated for a winning user is her 'critical payment', i.e., the minimum she needs to bid to obtain her requested bundle.

It is shown in [7] that the x_j values determined by solving the LP in line 7 are monotone with respect to the user valuations, i.e., a user u_j can increase her probability of winning by increasing her valuation. We now show that the randomized rounding step of CA-LP maintains the monotonicity of allocation. We can have two different cases in the randomized rounding step (lines 11-15),

$$\sum_{j:x_j>0} r_i^j \leq k_i, \forall i \in \{1,\ldots,m\} \tag{3.8}$$

or

$$\sum_{j:x_j>0} r_i^j > k_i, \exists i \in \{1,\ldots,m\} \tag{3.9}$$

Equation (3.8) represents the condition at which each user u_j having $y_j \leq x_j$ is guaranteed to get her requested bundle. Therefore, the probability of user u_j to be finally included in the set of winners is exactly x_j and the allocation is monotone.

On the other hand, when Equation (3.9) holds for some i, we divide the users into two groups as follows. First, let us assume that $x_1,\ldots,x_n$ are in decreasing order. Now, let l be the largest index for which the following equation holds.

$$\sum_{j\in\{1\ldots l\},x_j>0} r_i^j \leq k_i, \forall i \in \{1,\ldots,m\} \tag{3.10}$$

Therefore, a user $u_j, j \leq l$, will be included in the winners list with probability x_j, which in turn is monotone with respect to her valuation.

Now, a user $u_j, l < j \leq n$, will get her allocation with probability x_j if

$$r_i^j + \sum_{j'<j, u_{j'} \in W} r_i^{j'} \leq k_i, \forall i \in \{1, \ldots, m\} \tag{3.11}$$

i.e., there are enough resources available to fulfill user u_j's request after determining the winners among $u_1, \ldots, u_{j-1}$. Therefore, the probability of user u_j winning her bundle is given by:

$$Pr\left[r_i^j + \sum_{j'<j, u_{j'} \in W} r_i^{j'} \leq k_i, \forall i \in \{1, \ldots, m\}\right] x_j \tag{3.12}$$

The probability given by Equation (3.12) decreases as j increases (i.e., x_j decreases). User u_j can increase her probability of winning by reporting a higher valuation. Therefore, the allocation is monotone with respect to her valuation, although it is not directly proportional to x_j.

Considering the above two cases, we claim that the allocation algorithm of CA-LP determines the set of winners with a probability that is monotone with respect to the user valuations.

The payment calculated by CA-LP is the critical value that is the minimum a user must bid to get her requested bundle allocated. Her reported valuation only helps decide whether she will be a winner, but she has to pay this critical value when she wins, no matter how large her valuation is. Because of these and following the results given in [7] the CA-LP mechanism is truthful in expectation. □

Example 1. We show the execution of CA-LP for a small VMAP instance illustrated in Table 3.1. In this VMAP instance, six users are placing their bids and the cloud provider has two types of VM instances with eight available copies for each type of instance. Each row of the table represents a user. The first four columns list the user index j, the requested number of VM instances of type-1 (r_1^j) and type-2 (r_2^j), and the user's valuation (v_j). For example, user u_1's bid is $B_1 = (0, 4, 0.74)$ specifying a request for zero instances of type VM_1 and four instances of type VM_2, and a valuation of 0.74 for this bundle. Column x_j

Table 3.1: CA-LP Example

j	r_1^j	r_2^j	v_j	x_j	y_j	$u_j \in W$	p_j
1	0	4	0.74	0	0.43	N	0
2	3	4	7.62	0.85	0.32	Y	3.65
3	4	1	6.02	0.62	0.61	N	0
4	1	3	7.54	1	0.74	Y	2.01
5	2	1	5.94	1	0.14	Y	3.49
6	1	0	0.97	0	0.95	N	0

shows the fractional allocation values for each user computed by the LP. The next column is the random value (y_j) used to decide the allocation. We see that users $u_2, u_3, u_4,$ and u_5 have higher x_j than the corresponding y_js. But it is not possible to allocate the requested bundles to all these users, because that will exceed the number of available VMs of both types. Therefore, we first eliminate u_3 from the set of winners since x_3 is the minimum among these x_js. After this elimination, the set of winners satisfies all constraints. We show the final allocation decision in the column titled '$u_j \in W$', where 'Y' means the bundle is allocated and 'N' means the bundle is not allocated.

The values in the y_j column are also used in payment calculation. For example, the amount bidder u_4 will pay to the resource provider is determined by solving the LP with different valuations of u_4. Here, we perform a binary search between zero and 7.54 (i.e., v_4) to find out the valuation v_4' and solve the LP to find a new x_4', such that $x_4' < y_4$ (i.e., $x_4' < 0.74$) for valuations smaller than v_4' and $x_4' > y_4$ for valuations greater than v_4'. We find that for $v_4' = 2.0138$, $x_4' = 0.82 > 0.74$ and for $v_4' = 2.0129$, $x_4' = 0 < 0.74$. The search ends here by deciding the payment $p_4 = 2.0129$, which is shown rounded to two decimal digits in Table 3.1. We show the payment for all users in column p_j. Thus, users u_2, u_4 and u_5 obtain their requested bundles and pay 3.65, 2.01, and 3.49, respectively.

CA-GREEDY Mechanism

Lehmann *et al.* [35] proposed a $\sqrt{M}$-approximation mechanism for combinatorial auctions with single-minded bidders where the total number of items that need to be allocated is

Algorithm 3 CA-GREEDY Mechanism

{*Phase 1: Collect Bids*}
for $j = 1, \ldots, n$ **do**
  Collect bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$ from user u_j
end for
{*Phase 2: Winner Determination*}
$W \leftarrow \emptyset$;
for $j = 1, \ldots, n$ **do**
  $s_j \leftarrow \sum_{i=1}^{m} r_i^j w_i$
end for
re-order users such that
  $v_1/\sqrt{s_1} \geq v_2/\sqrt{s_2} \geq \ldots \geq v_n/\sqrt{s_n}$
for $j = 1, \ldots, n$ **do**
  if for all $i = 1, \ldots, m$, $r_i^j + \sum_{u_{j'} \in W} r_i^{j'} \leq k_i$ **then**
    $W \leftarrow W \cup u_j$
  end if
end for
{*Phase 3: Payment*}
for all $u_j \in W$ **do**
  $W_j' \leftarrow \{u_l : u_j \notin W \Rightarrow u_l \in W\}$
  $l \leftarrow$ minimum index in W_j'
  if $W_j' \neq \emptyset$ **then**
    $p_j \leftarrow (v_l/\sqrt{s_l})\sqrt{s_j}$
  else
    $p_j \leftarrow 0$
  end if
end for
for all $u_j \notin W$ **do**
  $p_j \leftarrow 0$
end for

M. We extend this mechanism by redefining M to be the weighted total number of VM instances, i.e., $M = \sum_{i=1}^{m} k_i w_i$. Here we define the 'size' s_j of the bundle in bid B_j requested by user u_j as $s_j = \sum_{i=1}^{m} w_i r_i^j$, while in the original mechanism, s_j is defined as $s_j = \sum_{i=1}^{m} r_i^j$, i.e., the total number of items requested in B_j. Our CA-GREEDY mechanism is given in Algorithm 3.

CA-GREEDY determines the winners by first ranking the users in decreasing order of their 'bid density' (i.e., $v_1/\sqrt{s_1}$) and then greedily allocating them starting from the top of

the list. Before allocating a new bundle the mechanism verifies that the new allocation does not exceed the number of available VM instances of each type (lines 11-15). The payment p_j a winner u_j pays is calculated by multiplying $\sqrt{s_j}$ with the highest bid density among the losing bidders who would win if u_j would not be a winner (lines 17-24). That is, the winner pays the critical value.

Our mechanism differs from the mechanism proposed by Lehmann *et al.* [35] in the way the bid density is calculated. The original mechanism computes s_j as the total number of items in B_j, while in our case we consider s_j to be the weighted sum of the number of VM instances requested in B_j. Another difference is in the way our mechanism verifies if the capacity is exceeded for each type of VM instance (line 12). These two differences are significant because the original problem formulation assumes that each item is of different type and that different types of items do not have any relative importance to the auctioneer. In our setting a cloud provider allocates different types of VM instances, which have different characteristics and are valued differently by the cloud provider. Thus, we associate a weight to each VM instance type in order to reflect these differences. In line 12, the original mechanism needs to check whether there is a common item in the bundle of the user that is currently being allocated and the ones that are already in the set of winners. Since there are lots of VM instances of the same type, we changed this and check if the number of instances of each type allocated to the winning bidders does not exceed the number of available instances of each type. We claim that CA-GREEDY has the same approximation ratio as the original greedy mechanism and it is truthful as well.

Theorem 2. *CA-GREEDY is a truthful mechanism that computes a $\sqrt{M}$-approximate solution to VMAP, where $M = \sum_{i=1}^{m} k_i w_i$.*

Proof. The mechanism proposed by Lehmann *et al.* [35] is an $\sqrt{M}$-approximation mechanism that solves the general combinatorial auction problem, where M is the total number of items. In the case of VMAP, $M = \sum_{i=1}^{m} k_i w_i$. According to the definition of $\mathbf{w}$, w_i is the number of VM_1 instances equivalent to one VM_i instance. Therefore, in VMAP, M is the

total number of equivalent instances of VM_1 that are available. The mechanism proposed by Lehmann *et al.* [35] provides a $\sqrt{M}$-approximation solution when there are M items in total, therefore the CA-GREEDY mechanism also generates an $\sqrt{M}$-approximation solution to the VMAP.

Now, we show that the winner determination algorithm of CA-GREEDY is monotone and the payment calculated for a winner is the critical value. From line 10 of the mechanism, it is clear that a user can increase her chance of winning by increasing her bid. Also, a user can increase her chance to win by decreasing the weighted sum of the items. For example, a user requesting two small and two large VM instances will be higher in the order than a user requesting one small and three large instances for the same valuation, although the numbers of VMs requested are the same. Therefore, the winner determination algorithm of CA-GREEDY is monotone with respect to user bids considering the relative computing capacities of different types of VMs. Finally, a winning bidder u_j pays the minimum amount she has to bid to win her bundle, i.e., her critical value. This is done by finding the losing bidder u_l who would win if u_j would not participate in the auction. User u_j's minimum bid density has to be at least equal to the bid density of user u_l for winning her bundle. Therefore, her critical valuation is $(v_l/\sqrt{s_l})\sqrt{s_j}$, which is the payment calculated by CA-GREEDY. Thus, the CA-GREEDY mechanism has a monotone allocation algorithm and charges the winning bidders their critical payment. We conclude that CA-GREEDY is a truthful mechanism. □

Example 2. In Table 3.2, we show the allocation and payment computation obtained by CA-GREEDY for the same instance of VMAP we used in Example 1. Here we also assume that $w_1 = 1$ and $w_2 = 2$, i.e., one instance of VM_2 is two times more powerful than one instance of VM_1. There are eight available instances of each type of VM. In Table 3.2, the first four columns represent the user index, the number of VMs of each type in their bundle and their valuation for that bundle. The value in the column titled 's_j' is the weighted sum of the total number of VM instances in a bundle. The next column, titled '$v_j/\sqrt{s_j}$', gives the relative valuation of users with respect to the weighted bundle size, that is the

Table 3.2: CA-GREEDY Example

j	r_1^j	r_2^j	v_j	s_j	$v_j/\sqrt{s_j}$	$u_j \in W$	p_j
1	0	4	0.74	8	0.26	N	0
2	3	4	7.62	11	2.3	N	0
3	4	1	6.02	6	2.46	Y	5.63
4	1	3	7.54	7	2.85	Y	0.69
5	2	1	5.94	4	2.97	Y	4.6
6	1	0	0.97	1	0.97	Y	0

'bid density'.

To determine the set of winners, we include users in descending order of $v_j/\sqrt{s_j}$ in the set of winners unless the inclusion violates the constraint that only eight copies of each type of VM can be allocated. User u_5 is the first winner and we allocate two copies of type VM_1 and one copy of type VM_2 to her. The next user to get an allocation is u_4, thus three instances of type VM_1 and four instances of VM_2 are allocated so far. Next, u_3 is selected for allocation, raising the total VM allocation to seven for VM_1 and five for VM_2. We see that the next user in the order is u_2, but allocating u_2 requires three instances of type VM_1 and four of type VM_2, whereas there is only one instance of type VM_1 and three instances of type VM_2 remaining. Therefore, u_2 is not included in the set of winners. User u_6, is next in the order not violating the constraints, thus she is included in the set of winners. So far, eight instances of VM_1 and five instances of VM_2 are allocated, leaving only three VM_2 instances not allocated. The last user, u_1, cannot obtain her allocation since she requests four instances of VM_2.

We show the payment calculation for user u_4 as an example. If u_4 is not a winner, there will be one and six instances of VM_1 and VM_2 to be allocated. The first non-winning user with respect to the order is u_2, but the number of VM instances available is not enough to allocate u_2. But the other remaining user, u_1's request can be fulfilled when u_4 is not a winner. Therefore, u_4's payment is calculated by multiplying u_1's bid density value by $\sqrt{7}$. Since no such user could be found for u_6, u_6's payment is zero.

Here, we note that the total revenue generated by CA-LP is 9.15 and that generated

by CA-GREEDY is 10.92. However, it is not guaranteed that CA-GREEDY will always generate higher revenue than CA-LP. The payment of the CA-LP mechanism depends on both the random variables y_j generated during the allocation phase and the competition among the bidders. On the other hand, the payment determined by the CA-GREEDY mechanism depends only on the competition among the bidders. In the example, we see that the highest four bid densities are between 2.3 and 2.97, where 2.3 is the bid density of user u_2, which is highest among the losing bids. Since this value is close to the winning bids, the winning bidders need to pay more to win their bundles. However, in a different scenario the CA-LP mechanism may generate higher revenues.

3.4 Experimental Results

We perform simulation experiments with different instances of VMAP. We solve these problems by employing the three mechanisms presented above. We compare the results and discuss the applicability of these mechanisms under different scenarios.

3.4.1 Experimental Setup

The simulation for one instance of VMAP runs for five simulation days. During each simulation, a maximum of $N = 100,000$ users are generated. Groups of users are created five times an hour, i.e., every twelve minutes. Therefore, an average of about 167 users are generated every twelve minutes. We add a deviation randomly chosen from [-20%, +20%] to this number to determine the actual number of users generated at a particular time. We invoke all three mechanisms every hour, with all the users generated during that hour and all users from previous time slots that are still active. An active user is one whose task has not been finished or the task deadline has not been reached. Each mechanism computes the allocation and pricing for the next one hour time-frame and keeps track of the users' status separately. We would like to emphasize here that each run of the mechanisms computes the allocation and payment of a given user for only one time slot and not for all the time

slots required for the user's task to complete execution. The user will need to participate in and win several auctions in order to complete her task.

Users are of three categories: type-1, type-2, and type-3. Type-1 users are the most demanding, type-3 the least, and type-2 users fall in between. User demands are characterized by four factors: number of requested VMs, valuation, duration for which the bundle is requested, and a deadline by which the task has to be finished. For example, type-1 users request more VMs than the other two types of users, request the VMs for longer periods of time, have the highest valuations, and have stricter deadlines than the others. Also, each category of users are generated at particular times of the day. A simulation day is divided into three periods: peak (8am–4pm), off-peak (4pm–midnight), and night (midnight–8am). Type-1 users are generated (and hence submit their bids) during the peak hours only. Type-2 users submit bids only during peak and off-peak hours while type-3 users submit bids at any time of the day. To compare with real life scenarios, we can roughly consider that type-1 users are the big corporations, type-2 are the large and medium businesses, and type-3 are the small businesses and individual users.

We assume that the cloud provider offers four types of VM instances: small, medium, large, and huge ($VM_1, VM_2, VM_3,$ and VM_4). We set their relative weights to $\mathbf{w} = (1, 2, 4, 8)$ and their fixed prices to $\mathbf{f} = (0.12, 0.24, 0.48, 0.96)$. This corresponds to the fixed-price model used in Microsoft's Windows Azure Platform [41]. We call this vectors a linear price vector since $f_i = 0.12 \cdot w_i$, for $i = 1, \ldots, 4$. Each user u_j's bid is a 5-tuple $(r_1^j, r_2^j, r_3^j, r_4^j, v_j)$, where r_i^j is the number of requested instances of VM_i and v_j is her valuation. User u_j's task is characterized by the tuple (t_j, d_j), where t_j is the duration for which the resources are requested and d_j is the time by which u_j's job needs to be completed.

To generate user bids, first the type of the user is randomly chosen from the user distribution. Then, random numbers are generated from the ranges $[r^{min}, r^{max}]$, $[0, v^{max}]$, and $[t^{min}, t^{max}]$ and assigned to r_i^j, v_j, and t_j, respectively. These values are then scaled with a factor associated with the category of user. For example, the scale factors for r_i^js are given by the vector $\boldsymbol{\rho}$. Therefore, after generating r_i^j values from the given range, they are

Table 3.3: Simulation Parameters

Parameter	Description	Value(s)
m	Types of VMs	4
$k_1, \ldots, k_m$	Available VMs of each type	500, 1000, 2000
$\mathbf{w}$	Relative weight of VMs	(1, 2, 4, 8)
$\mathbf{f}$	Fixed-price vector	(.12, .24, .48, .96) (.12, .22, .39, .70) (.12, .26, .58, 1.28)
$\boldsymbol{\phi}$	Fixed-price factor vector	(1, 1, 1) (3, 2, 1) (4, 2, 1)
N	Maximum number of users	100000, 50000, 10000
n	Number of users in an auction	Varies
r^{min}, r^{max}	Min. & Max. VM instances of each type in a bundle	0, 5
v^{max}	Maximum valuation	1, 2, 5, 10
t^{min}, t^{max}	Min. & Max. execution time	1, 10
d^{min}, d^{max}	Min. & Max. deadline	2, 10
$\boldsymbol{\pi}$	Distribution of users	(10%, 40%, 50%), (20%, 30%, 50%), (20%, 40%, 40%), (30%, 30%, 40%)
$\boldsymbol{\rho}$	Scale factor for bundle size	(2, 1.5, 1), (3, 2, 1)
$\boldsymbol{\lambda}$	Scale factor for valuation	(2, 1.5, 1), (3, 2, 1)
$\boldsymbol{\tau}$	Scale factor for execution time	(2, 1.5, 1), (3, 2, 1)
$\boldsymbol{\delta}$	Scale factor for deadline	(0.5, 0.67, 1), (0.33, 0.5, 1)

multiplied by ρ_1 to determine the actual value when the user is of type-1. To illustrate this, suppose we generate some $r_i^j = 5$ for user u_j and $\boldsymbol{\rho} = (2, 1.5, 1)$. Now, the actual r_i^j value of users of type-1, type-2, and type-3 will be $5 \times 2 = 10$, $5 \times 1.5 \approx 8$, and 5, respectively. Similarly, the elements of vector $\boldsymbol{\lambda}$ give the scaling factors for valuation of different types of users. After generating a random number within the range $[0, v^{max}]$, we multiply it with the entry in λ corresponding to the type of the generated user. Similarly, vectors $\boldsymbol{\tau}$ and $\boldsymbol{\delta}$ denote the factors for scaling the time required and the deadline. The deadline is

determined by selecting a random number, scaling it, and then adding the result to t_j. We list all simulation parameters in Table 3.3.

To create different instances of VMAP, we vary the parameters that affect the user distribution, demand, and payment resulting from the allocation. Thus, we choose four different distributions of type-1, type-2, and type-3 users given by the following tuples: (10%, 40%, 50%), (20%, 30%, 50%), (20%, 40%, 40%), and (30%, 30%, 40%). We consider four values of v^{max}, 1, 2, 5, and 10, which give four ranges of valuations (0-1), (0-2), (0-5), and (0-10). We also vary the number of available VM instances and the factors that distinguish bids of different types of users. Table 3.3 lists the parameters, their description, and the range of values they take. Combining all these values with each other, our simulation experiment simulated 768 different instances of VMAP.

In addition to the above set of experiments, we perform six sets of experiments with 768 VMAP instances each – by varying only one of the parameters listed above. We create two sets of such experiments by setting N, the maximum number of users to 50,000 and 10,000, respectively. From these experiments we try to evaluate the VM allocation mechanisms for various degrees of user demands. In the next two sets of experiments, we set $N = 100,000$ and consider two different fixed-price vectors $\mathbf{f}$ as follows. A sublinear price vector with prices for instance VM_i given by $f_i = 0.12 \cdot (w_i)^{0.85}$, which corresponds approximately to $\mathbf{f} = (0.12, 0.22, 0.39, 0.70)$; and a superlinear price vector with prices for instance VM_i given by $f_i = 0.12 \cdot (w_i)^{1.15}$, which corresponds to $\mathbf{f} = (0.12, 0.26, 0.58, 1.28)$. Since the FIXED-PRICE mechanism heavily depends on the fixed prices of the VM instances, these experiments let us determine whether the fixed-price vector affect the performance of the proposed mechanisms.

Finally, we vary the fixed-price vectors during the peak and off-peak hours of the day to examine whether they can generate higher revenue by capturing the higher demands during these times. This is accomplished by introducing the *fixed-price factor vector* $\boldsymbol{\phi}$. This is a 3-vector containing factors that are used as multipliers for the fixed-price vector during different hours of the day. For example, $\boldsymbol{\phi} = (3, 2, 1)$ indicates the fixed prices of each type of VM

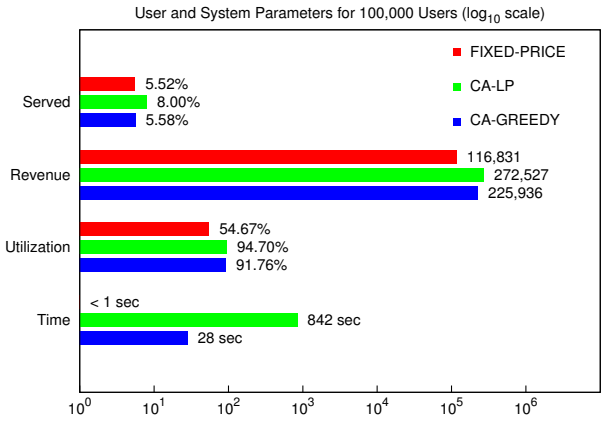


Figure 3.2: Overall performance of the mechanisms with linear fixed-price vector (.12, .24, .48, .96), fixed-price factor vector $\boldsymbol{\phi} = (1, 1, 1)$, and 100,000 users. The plot is drawn at $\log_{10}$ scale.

instance will be multiplied by 3 during the peak hours, by 2 during the off-peak hours, and by 1 during night hours. If the fixed price-vector is $\mathbf{f} = (0.12, 0.22, 0.39, 0.70)$ then the prices for the four types of VM instances during peak hours are given by (0.36, 0.66, 1.17, 2.1). In the regular case, $\boldsymbol{\phi} = (1, 1, 1)$, that is, the prices for VM instances are the same for all periods of the day. In our experiments we use two price factor vectors (3, 2, 1) and (3, 2, 1), that is, we consider that during peak hours the prices are four, and respectively three times higher than during the night hours, while the prices during off-peak hours are two times higher than the prices during night hours. This will allow us to investigate the effect of taking into account the demand when establishing prices for the fixed-price mechanisms. Table 3.3 lists these price vectors.

3.4.2 Analysis of Results

The experimental results show that the proposed combinatorial auction-based mechanisms have clear advantages over the fixed-price mechanism for solving the VMAP. Here we discuss their overall performance and then investigate the effect of different parameters on various performance metrics such as generated revenue, utilization, runtime, and the number of

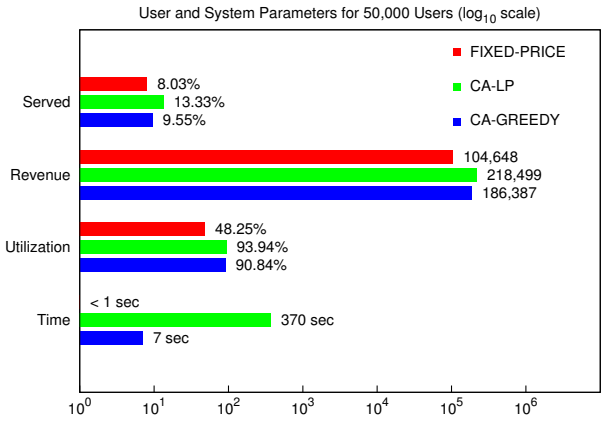


Figure 3.3: Overall performance of the mechanisms with linear fixed-price vector (.12, .24, .48, .96), fixed-price factor vector $\phi = (1, 1, 1)$, and 50,000 users. The plot is drawn at $\log_{10}$ scale.

users served by the system.

First, we present the average performance of the mechanisms in Figures 3.2 to 3.5. All the plots in these figures are represented using a logarithmic scale. The fixed price mechanism used in these experiments assumes the same fixed price for all the periods of the day, that is, the fixed price factor vector $\phi = (1, 1, 1)$. In Figure 3.2 we present the summary of the experiments with 100,000 users and the linear price vector. We see that CA-LP outperforms the other two mechanisms in all the metrics except the running time. Here the running time is the average time needed to run one auction simulation. About 8% of the 100,000 users could complete their tasks while running the CA-LP mechanism. We also see that the overall utilization of the resources and the revenue generated are the best for CA-LP. This is because the linear program has as objective maximizing the sum of the valuations, which eventually generates higher revenue by utilizing as many machines as possible while satisfying the constraint given in Equation (3.6). Utilizing more machines allocates more users and therefore more users can finish their tasks. On the other hand, the CA-GREEDY mechanism allocates users based on their relative valuation. Therefore, it cannot always utilize resources as much as CA-LP can. But the running time of the CA-LP

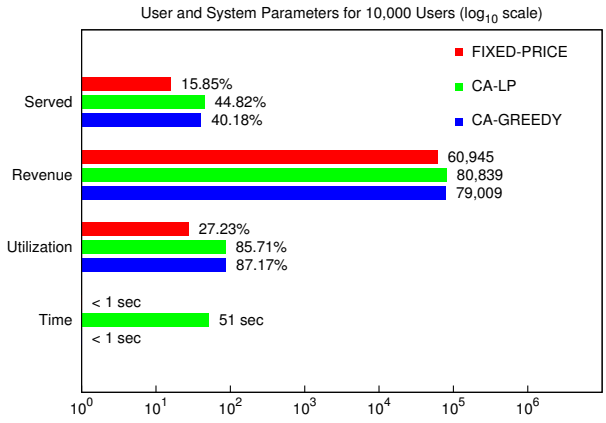
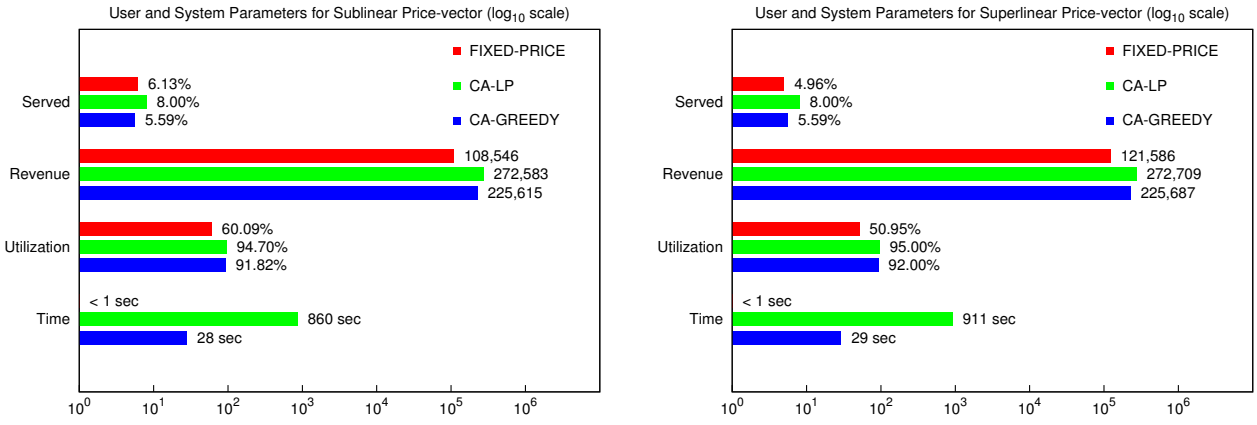


Figure 3.4: Overall performance of the mechanisms with linear fixed-price vector (.12, .24, .48, .96), fixed-price factor vector $\phi = (1, 1, 1)$, and 10,000 users. The plot is drawn at $\log_{10}$ scale.

is prohibitively high because the payment calculation involves repeated solving of the linear program. The FIXED-PRICE mechanism obviously has the lowest running time because it only allocates users on a first-come, first-served basis. The CA-GREEDY mechanism has very low running time compared to CA-LP since its only major computation is to sort the list of users.

In Figures 3.3 and 3.4, we show the summary of the results for experiments with 50,000 and 10,000 users and linear price vector. First, we observe that each of the mechanisms serves higher percentage of users, generates lower revenue, and utilizes less resources as the number of users decreases. This trend with decreasing demand is natural for any allocation mechanism. We further observe that the rank of the mechanisms in terms of all the metrics remain the same regardless of the total number of participants. Also note that compared to the FIXED-PRICE mechanism, the increase in served users is much higher for CA-LP and CA-GREEDY. This is due to the fact that FIXED-PRICE only considers those users who bid at least the fixed value, while the auctions determine allocations based on the market demand and supply. For the same reason, the utilization of the machines decreases at a slower rate in the case of combinatorial auction-based mechanisms than in the case of



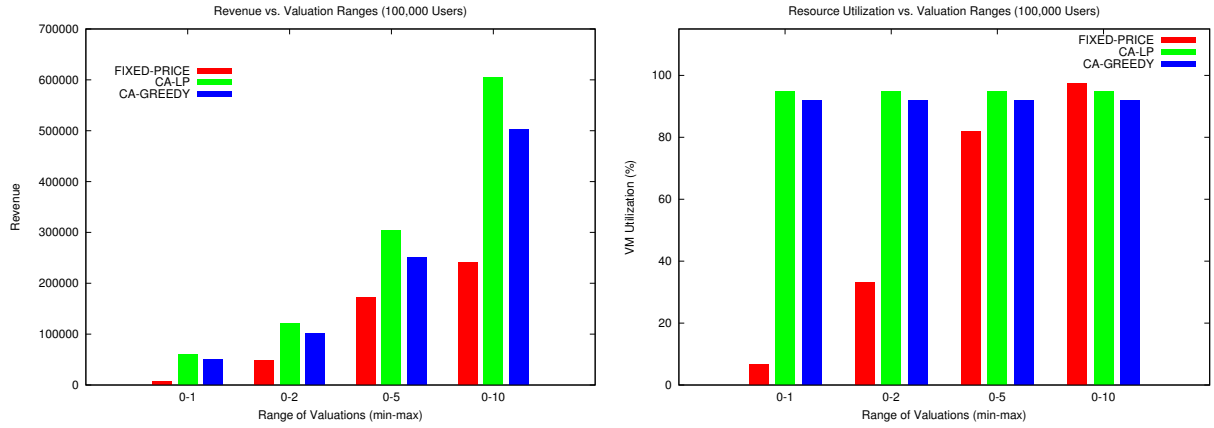
(a) (b)

Figure 3.5: Overall performance of the mechanisms with 100,000 users and (a) sublinear fixed-price vector (.12, .22, .39, .70); (b) superlinear fixed-price vector (.12, .26, .58, 1.28). The fixed-price factor vector $\boldsymbol{\phi} = (1, 1, 1)$. The plot is drawn at $\log_{10}$ scale.

the fixed-price mechanism. However, the gap between the total revenue generated reduces when there are less participants, as the auction-based mechanisms generate less revenue when there is less competition.

In Figures 3.5a and 3.5b, we summarize the results of the experiments with 100,000 users and sublinear and superlinear fixed-price vectors, respectively. By comparing them with the results in Figure 3.2, we see that the only mechanism affected is FIXED-PRICE, which can serve more users and utilize more resources when the price vector is sublinear. However, in this case the total revenue decreases as users pay less than what they pay in the case of a linear price vector. Naturally, we see the opposite trend for the superlinear price vector. We can conclude that we cannot improve the overall quality of the allocation generated by the FIXED-PRICE mechanism. By changing the price vector we can only improve one metric while sacrificing another.

We now investigate different performance metrics by varying other simulation parameters, while setting the total number of users to 100,000 and using the linear price vector. In Figure 3.6a, we show the revenue generated for different ranges of user valuations. We

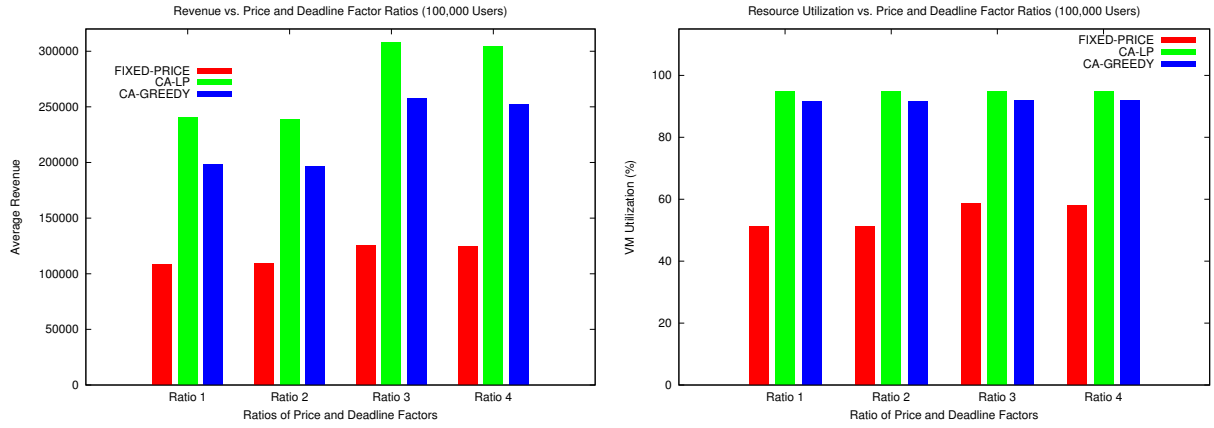


(a) (b)

Figure 3.6: Effect of valuation ranges (with 100,000 users) on (a) Revenue; (b) VM utilization.

see that low user valuations most adversely affect the FIXED-PRICE mechanism. This is because it does not allocate the requested bundles to users having valuations below the fixed-price range. On the other hand the combinatorial auction mechanisms can generate higher revenues because they determine the payments from the user valuations. The revenue increases at the same rate from valuation ranges (0–1) to (0–5). Then, for the valuation range (0–10), we see a sharp rise in revenue generated by the auction mechanisms, while the FIXED-PRICE mechanism's revenue does not increase that much. This is because the price for an average-sized bundle is 4.5 according to the fixed prices we set. FIXED-PRICE mechanism's revenue is bounded by the fixed prices, therefore it cannot take advantage of higher user valuations. As shown in Figure 3.6b, our experiments reveal that the rate of resource utilization obtained by the auction-based mechanisms is not affected by the valuation ranges. The utilization obtained by the FIXED-PRICE mechanism increases as the valuation range increases, and it is lower than that obtained by combinatorial auction mechanisms for all the ranges of valuations except for the (1–10) range.

In Figure 3.7, we show the average revenue and resource utilization generated by the mechanisms when different values of the scale factors for valuation ($\boldsymbol{\lambda}$) and deadline ($\boldsymbol{\delta}$) are used. As a reminder, a scale factor for valuation (or, the price factor) represents how



(a) (b)

Figure 3.7: (a) Revenue and (b) VM utilization vs. ratios of price and deadline factors. Ratio is defined as a set of ((price-factor), (deadline-factor)) values. Ratio 1 = ((2, 1.5, 1), (.33, .5, 1)), Ratio 2 = ((2, 1.5, 1), (.5, .67, 1)), Ratio 3 = ((3, 2, 1), (.33, .5, 1)), Ratio 4 = ((3, 2, 1), (.5, .67, 1)).

much more a bundle is valued by a type-1 and type-2 user than a type-3 user. For example, $\boldsymbol{\lambda} = (2, 1.5, 1)$ denotes the case where on average a type-1 user bids twice the value than a type-3 user and a type-2 user bids around 1.5 times higher than a type-3 user. When $\boldsymbol{\lambda} = (3, 2, 1)$, these multiplication factors become 3 and 2, respectively and meaning that those users' demands are even higher than those of type-3 users. Similarly, a deadline factor says how strict is the deadline of type-1 and type-2 users compared to that of type-3 users. We consider four possible combinations of these two factors, which we denote as Ratio 1, ..., Ratio 4 in Figure 3.7. We show the revenue generated in different such scenarios in Figure 3.7a. Ratios 1 and 2 are for the price factors (2, 1.5, 1) and Ratios 3 and 4 are for the price factors (3, 2, 1). We see that the combinatorial auction-based mechanisms are capable of generating higher revenues when the type-1 and type-2 bidders bid more, but the fixed-price mechanism cannot increase the generated revenue that much. However, we see that deadline factors do not have much effect on the outcome, as evident from similar values shown for different deadline factors but the same valuation factor (e.g., Ratio 1 and Ratio 2). From Figure 3.7b we see that these factors have almost no effect on machine

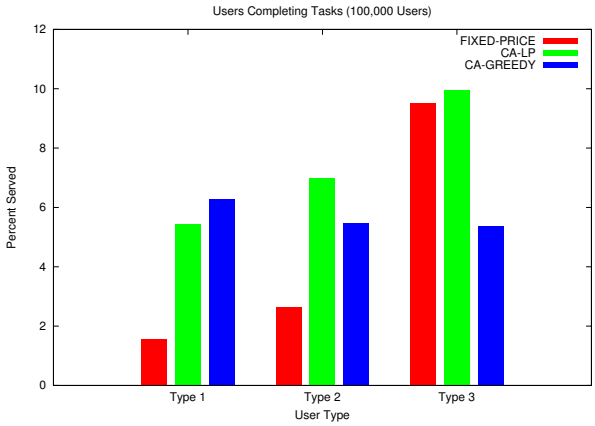


Figure 3.8: Percentage of served users for simulations with 100,000 users.

utilization achieved by the combinatorial auctions. But utilization is increased a little for the FIXED-PRICE mechanism with higher valuation factors.

We now examine how the three mechanisms deal with different types of users. Recall that type-1 users are the most demanding and type-3 users are the least demanding. First, we show the percentage of users who could complete their tasks in Figures 3.8 to 3.10. We refer to these users as the served users. In these figures we show the results from three different sets of experiments, with 100,000, 50,000, and 10,000 users and the linear price vector. In Figure 3.8, where the total number of users is 100,000, we observe that the FIXED-PRICE mechanism serves type-3 users the most. This is because it only considers the order in which users arrive. Type-1 and type-2 users have shorter deadlines and therefore leave the system if they do not get the allocation within a few allocation events. On the other hand, type-3 users have longer deadlines, and therefore they are active longer and eventually get the allocation once the users that entered the system earlier finish their tasks. CA-LP also served more users of type-3 than users of other types, yet it served more users compared to the FIXED-PRICE mechanism in every category. Here we see a nice property of CA-GREEDY that is, it serves more type-1 users than the other mechanisms. It also serves more users of type-1 than other user types. This is because CA-GREEDY makes decisions based on the bid densities, which are on average the highest for type-1

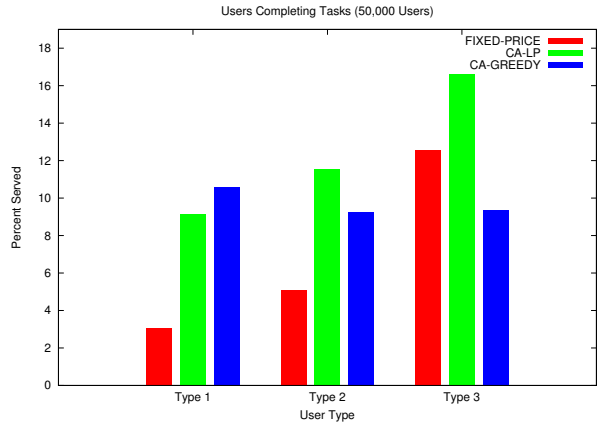


Figure 3.9: Percentage of served users for simulations with 50,000 users.

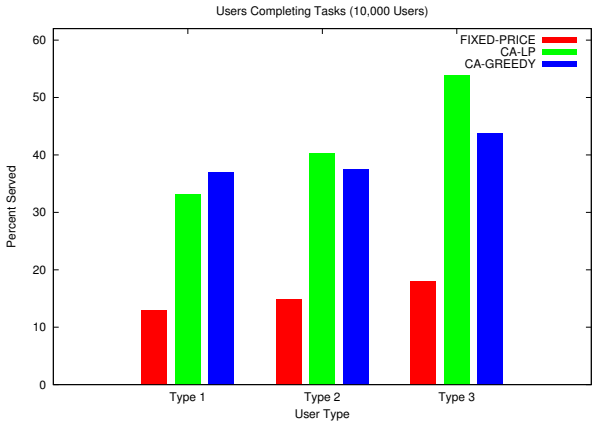


Figure 3.10: Percentage of served users for simulations with 10,000 users.

users.

Now comparing Figure 3.8 with Figures 3.9 and 3.10, we see that for 50,000 users the CA-GREEDY mechanism maintains its feature of serving a higher percentage of more demanding users than less demanding users. But for 10,000 users, the percentage of served users of type-3 is higher than the ones corresponding to the other two types of users. This is because for the reduced demand, type-1 and type-2 users cannot occupy most of the resources as they do for the cases with higher number of users. Also, recall that type-1 and type-2 users request larger bundles and are active during peak hours and off-peak hours only. On the other hand, type-3 users are generated any time of the day. Therefore, the

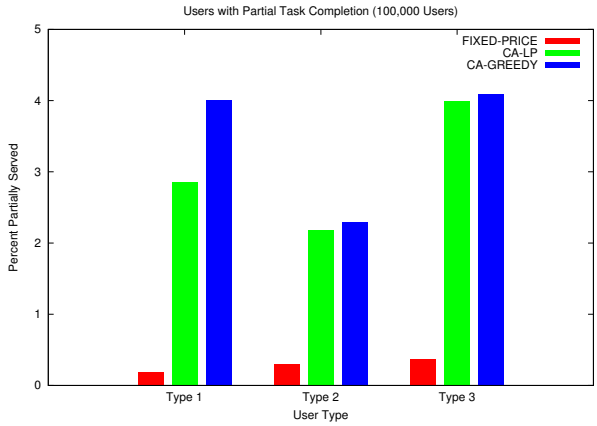


Figure 3.11: Percentage of partially served users for simulations with 100,000 users.

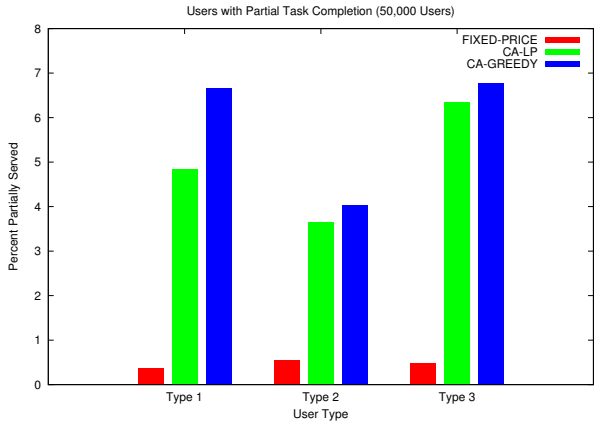


Figure 3.12: Percentage of partially served users for simulations with 50,000 users.

type-3 users get more space for occupying the resources facing less competition from the other users. Also, their bundle size is smaller compared to the other users and therefore a higher number of users can be served using the same amount of resources. On the other hand, in the case of CA-LP, we see almost the same trend (although at a different scale) in terms of serving the three types of users. Hence, we can conclude that CA-GREEDY is a better choice in terms of fairness and the handling of demand and supply in the market.

In Figures 3.11 to 3.13 we plot the percentage of users that could only partially complete their tasks. First, we observe that the FIXED-PRICE mechanism has the least number of partially served users in all three cases. This is because of its inherent first-come, first-

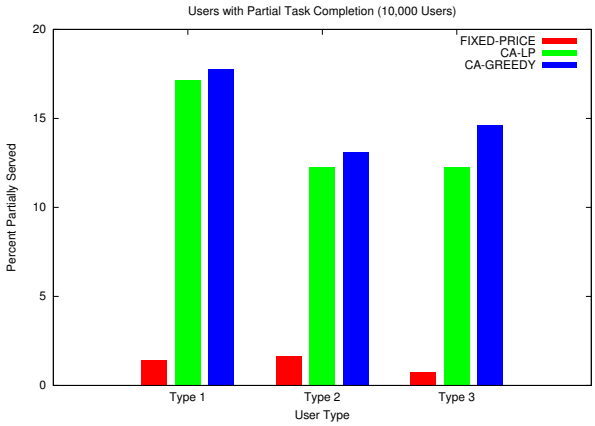


Figure 3.13: Percentage of partially served users for simulations with 10,000 users.

served policy. These plots also reveal that the percentage of partially served users increases in the case of CA-LP and CA-GREEDY when fewer users participate. Such effect is natural to auction mechanisms; a user must be denied the resources once another user with a higher bid arrives in an already saturated market for resources. However, if the consequences of having jobs partially completed is very important in some systems, it may be better to consider FIXED-PRICE as the allocation mechanism. The combinatorial auction-based mechanisms we propose can also be improved by incorporating some penalty for partially finished jobs. In the simulations we consider that the bids are generated once and a user submits the same bid until she gets her job done or her deadline is reached. In practice, a user is an interactive entity and can adapt her bid depending on the value and urgency of her job and the current market demand. Creating an automated bidding agent to participate in the combinatorial auctions could also be an interesting research direction that could eventually decrease the number of partially served users.

Now we present the average resource utilization obtained by the three mechanisms during different periods of time of the day. Recall that in the experiments, we divided a day between peak (8am-4pm), off-peak (4pm-midnight), and night (midnight-8am) hours. Also, type-1 users are generated during the peak hours, type-2 users during the peak and off-peak hours, and type-3 users can place their bids any time of the day. In Figure 3.14,

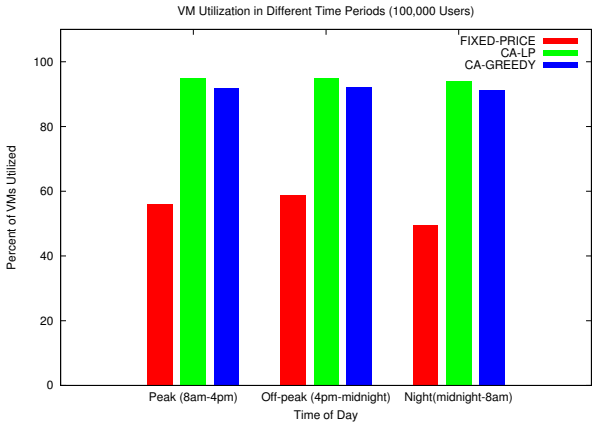


Figure 3.14: Utilization of resources during different periods of time (100,000 users)

we see that the resource utilization is around 95% for CA-LP for each period of the day. The utilization achieved by CA-GREEDY is very close to that of CA-LP for all three periods. The proposed mechanisms are able to effectively balance the load of the system over time. The utilization obtained by FIXED-PRICE is about 56% and 59% during peak and off-peak hours and it falls below 50% during night. Since FIXED-PRICE is a first-come-first-served mechanism, it cannot free up resources that are being used by type-3 users when in the morning type-1 users start placing their requests. Since type-1 users have shorter deadlines, by the time some resources are freed up, some users have already left the system. Therefore, the utilization obtained by FIXED-PRICE is far below that obtained by CA-LP and CA-GREEDY. At night, the utilization further drops because only the type-3 users request computing resources.

All the above experiments considered the FIXED-PRICE mechanism with a fixed-price factor vector $\boldsymbol{\phi} = (1, 1, 1)$. We now show the results obtained by considering different fixed-price factor vectors, $\boldsymbol{\phi}$. This is equivalent to considering different prices at different times of the day and it will allow us to investigate the effect of increasing the prices during high demand hours on the performance of the mechanisms. The combinatorial auction-based mechanisms dynamically determine the prices of the VM instances. Since demand varies during peak, off-peak, and night hours, we multiply the fixed prices with different values

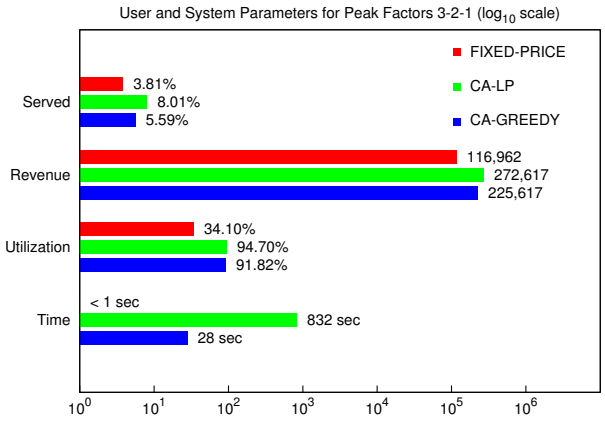


Figure 3.15: Overall performance of the mechanisms with fixed-price factor vector $\boldsymbol{\phi} = (3, 2, 1)$ and 100,000 users. The plot is drawn at $\log_{10}$ scale.

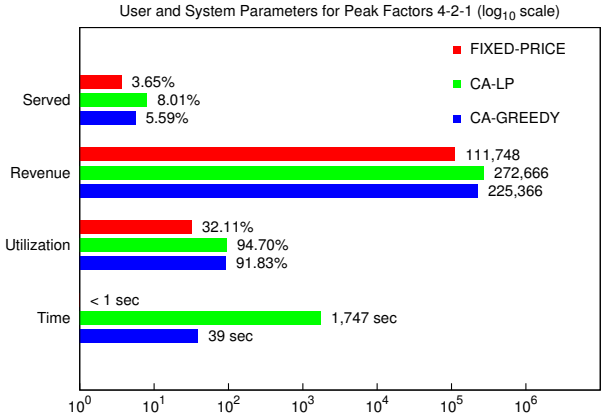


Figure 3.16: Overall performance of the mechanisms with fixed-price factor vector $\boldsymbol{\phi} = (4, 2, 1)$ and 100,000 users. The plot is drawn at $\log_{10}$ scale.

based on the time of the day. The results presented in Figure 3.15, are obtained when we multiply the fixed-price vector by 3 for the peak hours, by 2 for the off-peak hours, and by 1 for the night hours. This correspond to a fixed-price factor vector $\boldsymbol{\phi} = (3, 2, 1)$. We see that when compared to the results presented in Figure 3.2, the percentage of served users when employing the FIXED-PRICE mechanism has decreased from 5.5% to 3.8% and the utilization of VM instances has decreased from 54% to 34%. This is expected, since more users are being rejected allocation during peak and off-peak hours due to the increase in the

fixed-prices. However, the revenue generated by the FIXED-PRICE mechanism remains at almost the same level. This shows that adjusting the fixed-price vector in anticipation of higher demand may not improve much the overall efficiency. At higher prices, fewer users are served and resources are under-utilized leading to no significant impact on the generated revenue. Figure 3.16 shows the results obtained when we changed the fixed-price factor vector to $\boldsymbol{\phi} = (4, 2, 1)$, i.e., we multiply the fixed-price vector by 4 during the peak hours, by 2 during the off-peak hours and by 1 during night hours. Here we observe that while serving fewer users and utilizing less resources, the FIXED-PRICE mechanism also obtains lower revenue. We conclude that it is possible to control the behavior of fixed-price mechanisms by updating the fixed-price vector based on observation or statistical data about the demands. But combinatorial auction-based mechanisms compute the price dynamically, therefore no matter how the demand changes, they are able to obtain an efficient allocation and pricing.

In summary, we can conclude that combinatorial auction-based allocation and pricing mechanisms are more desirable over the fixed-price based ones currently employed by cloud providers. CA-LP is a better choice when the objective is to obtain higher revenue and higher utilization of resources. However, we have to limit the application of CA-LP to systems with small number of users, because otherwise the execution time will be prohibitive. This is because the CA-LP involves solving a linear program whose number of unknowns increases with the number of users participating in the auction. In addition to this, in order to compute the user's payments, CA-LP needs to solve one linear program for each user, thus the execution time increases very fast with the number of users. CA-LP can be a good choice when auctions are run at longer intervals. On the other hand, CA-GREEDY can be applied to cloud systems with any number of users being able to generate high revenue and resource utilization with very low execution time. CA-GREEDY is a better choice when the objective of VMAP is to maximize the social welfare. It is also worth mentioning that the CA-LP mechanism is designed for bidders with known bundles (i.e., bundles that are known to the auctioneer) [7]. Therefore, this mechanism is vulnerable to manipulation by

users who bid for unknown bundles in the hope of obtaining a better allocation or price. On the other hand, CA-GREEDY is a truthful mechanism with respect to both valuations and the bundles requested. Considering all the above aspects, we recommend using the CA-GREEDY mechanism for solving general purpose VM allocation problems in clouds.

3.5 Summary

We investigated the applicability of combinatorial auction-based mechanisms for allocation and pricing of VM instances in cloud computing platforms. We proposed two combinatorial auction-based mechanisms for solving the problem of allocating VM instances in clouds. We compared their performance with that obtained by a currently used fixed-price mechanism. We performed extensive simulation experiments and conclude that combinatorial auction-based mechanisms are clearly a better choice for VM allocation in clouds. Based on experimental data and on the theoretical properties of the mechanisms, we also made recommendations that the CA-GREEDY mechanism should be the choice for general purpose VM instance allocation problems while the CA-LP mechanism can be reserved for special scenarios.

CHAPTER 4: EFFICIENT BIDDING FOR VIRTUAL MACHINE INSTANCES IN CLOUDS

4.1 Introduction

In the previous chapter, we showed that combinatorial auctions are efficient mechanisms for allocating the VM instances in clouds. In a combinatorial auction-based VM instance allocation mechanism, a user bids for a bundle of VM instances of different types required for executing her application by specifying the bundle and a value. The value represents how much the user is willing to pay for the bundle if allocated. The auction mechanism determines the set of winning users, allocates their requested bundles, and computes the amount they have to pay. The goal of the cloud provider is to efficiently allocate the available VM instances to the users and generate the maximum possible revenue. On the other hand, a user desires to maximize her own *utility*, that is, the value she derives from obtaining the bundle minus the amount she has to pay for using the bundle. If the user does not obtain any bundle, her utility is zero. The combinatorial auction mechanisms we designed in the previous chapter are *incentive-compatible*, that is, they guarantee that a user maximizes her utility by bidding her 'true' valuation of the requested bundle. Therefore, to achieve maximum utility, a user needs to determine (i) the bundle of VM instances that guarantees the best performance for her application, and (ii) the correct (or true) valuation of the bundle.

However, generating such an efficient bid is a nontrivial problem, especially if the user

needs to execute parallel applications on the cloud platform. The degree of parallelism of the application and the system's parameters, such as communication delay, limit the speedup of a parallel application. Thus, it is imperative to incorporate these parameters when determining the best VM bundle necessary for executing an application. A realistic valuation of a given bundle of VM instances should also consider the performance gain of the application achieved by the bundle. Finally, the bid should not exceed the budget of the user.

4.1.1 Our Contribution

We address the above problem by first designing a user's valuation function that considers both the application and the system's parameters. The proposed valuation function determines the value of a bundle of VM instances on which a user executes her application. Then, we propose an algorithm that uses this valuation function to generate efficient bids within the users' budgets. We analyze the complexity of our proposed bidding algorithm and perform extensive simulation experiments to investigate its properties.

4.1.2 Organization

The rest of the chapter is organized as follows. In Section 4.2, we describe our proposed bidding strategy algorithm and analyze its complexity. In Section 4.3, we evaluate the proposed bidding strategy by extensive simulation experiments. We summarize our results in Section 4.4.

4.2 Proposed Bidding Strategy

In this section, first we briefly describe the setup of the combinatorial auction for VM instances allocation in clouds and then present our proposed efficient bidding strategy algorithm.

Among the combinatorial auction-based mechanisms that we designed for VM allocation in clouds, we choose CA-GREEDY to illustrate the design of our bidding strategy. The bidding strategy is not designed specifically for this mechanism and it is valid for any mechanism that solves the same VM allocation problem solved by CA-GREEDY. In Chapter 3, we showed that CA-GREEDY guarantees *incentive-compatibility* i.e., users obtain maximum utility only by bidding their true valuations for the requested bundles. Following is a quick refresher of the VM allocation problem that we solve using CA-GREEDY.

A cloud provider needs to allocate $k_1, \ldots, k_m$ copies of m different types of VM instances among n competing users. The different types of VMs are represented as $VM_1, \ldots, VM_m$ and their relative 'computing powers' are denoted by $\mathbf{w} = (w_1, \ldots, w_m)$, where w_i is the number of processors in VM_i. We assume that the VM instances have other computing resources (e.g., memory, storage, and bandwidth) in proportion to the number of processors. Here we assume that $w_1 = 1$ and $w_i \leq w_{i+1}$. An example is $\mathbf{w} = (1, 2, 4, 8)$ that means the cloud provider offers four types of VM instances having 1, 2, 4, and 8 processors, respectively.

A user u_j, $j = 1, \ldots, n$ participates in the auction for VM instances by submitting her bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$. Here, r_i^j is the number of VM_i instances that user u_j requests and v_j is the maximum amount that u_j is willing to pay if the bundle is allocated to her. The goal of the CA-GREEDY mechanism is to determine an allocation vector $\mathbf{x} = (x_1, \ldots, x_n)$, where $x_j = 0$ indicates that user u_j does not obtain her requested bundle, and $x_j = 1$ means that u_j obtains her requested bundle. The mechanism also determines a payment vector $\mathbf{y} = (y_1, \ldots, y_n)$, where y_j is the amount user u_j has to pay for the allocated bundle.

The combinatorial auction for VM instances is conducted periodically (e.g., once per hour). Each auction allocates the maximum possible number of resources to the users for only one period of time. Users who did not complete their application execution in one period need to bid again for the next period. It is the users' responsibility to determine how many units of time they need certain resources and submit their bids until their work is completed. Now we are ready to present the bidding strategy algorithm that we design

in this work.

A *bidding strategy* should create efficient *bids* for users participating in combinatorial auctions for VM instances in clouds. The strategy must take into account the type and workload of the application the user intends to run, and her budget. It should generate a bundle of VM instances and a *valuation* of that bundle in order to provide the user with the best performance within her budget.

4.2.1 Execution Time and Speedup

We consider a set of users who intend to run parallel applications on clouds. In particular, we consider *malleable applications*, which are parallel applications that can be executed on any given number of processors [32]. However, by employing more processors we cannot gain speedup indefinitely. This is because by adding more processors the overhead added to the execution time increases. This overhead is a function of both system-specific and application-specific parameters, as we will discuss in the following.

Let us consider that a user u_j needs to run a malleable parallel application a_j on a cloud platform. The application is characterized by its workload ω_j, expressed as a processor-time product. Havill and Mao [32] modeled the execution time T_j of an application a_j as:

$$T_j = \omega_j / p_j + (p_j - 1)\theta \tag{4.1}$$

where p_j is the number of processors used, and θ is a system-specific parameter called the 'setup time'. The setup time is the time spent to create, dispatch, or destroy multiple processes. The first term represents the time spent executing the workload only, which is equally divided to p_j processors. The second term corresponds to the system-specific overhead (e.g., time to create or destroy a process).

We argue that the overhead of execution on multiple processors is not only dependent on the setup time but also on the time spent on distributing the input data during initialization and on the time spent on communication and synchronization during program

execution. We assume that the amount of information to be exchanged during initialization and execution can be expressed as a fraction of the total workload. Hence, our proposed model for the *parallel execution time* of a malleable application is given by

$$T_j = \omega_j/p_j + (p_j - 1)\theta + (p_j - 1)\delta\tau_j\omega_j + (R_j - 1)\delta c_j\omega_j \tag{4.2}$$

Our new model adds two new terms to the existing two terms of Equation (4.1). The third term in the new equation for T_j characterizes the initialization overhead. Here, δ is the time to communicate one unit of data between different VM instances in the system, and τ_j is the amount of input data to be transferred, expressed as a fraction of the total application workload. Therefore, the amount of input data that is transferred is $\tau_j\omega_j$ and the overhead for data transfer during initialization is $(p_j - 1)\delta\tau_j\omega_j$. The fourth term represents the communication and synchronization overhead incurred during processing. Here, c_j is the communication overhead expressed as a fraction of the workload. R_j is the number of VM instances the total number of processors are divided into. Here we assume that communication within a VM instance takes negligible amount of time compared to the communication between VM instances. Therefore, we multiply the amount of communication by $R_j - 1$. This implies that if all p_j processors reside on one VM instance (i.e., $R_j = 1$), the overhead for communication and synchronization is negligible. Otherwise, it increases with the number of VM instances employed in running the application.

Using Equation (4.2) we derive the *speedup* of the application as:

$$S_j = \frac{\omega_j}{\omega_j/p_j + (p_j - 1)\theta + (p_j - 1)\delta\tau_j\omega_j + (R_j - 1)\delta c_j\omega_j} \tag{4.3}$$

We examine the characteristics of this speedup function in Figures 4.1 and 4.2. In both figures, the following parameters are fixed: $\omega_j = 50$, $\delta_j = 0.01$, $\tau_j = 0.05$, $\theta_j = 0.05$, $c = 0.01$. In Figure 4.1, we plot the speedup vs. the number of processors while keeping $R_j = 1$ constant. Here we see that the speedup improves until the number of processors

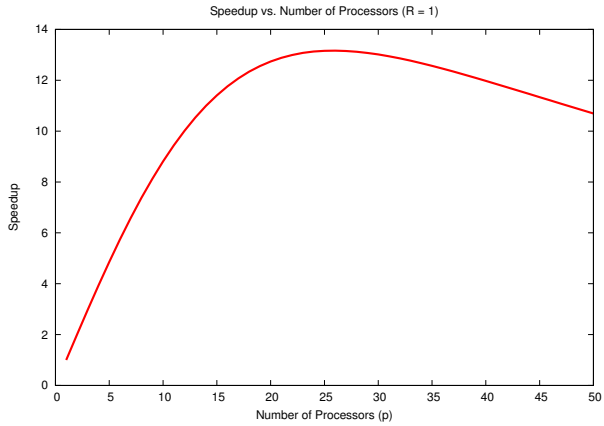


Figure 4.1: Characteristics of the valuation function: Speedup vs. number of processors ($R_j = 1$)

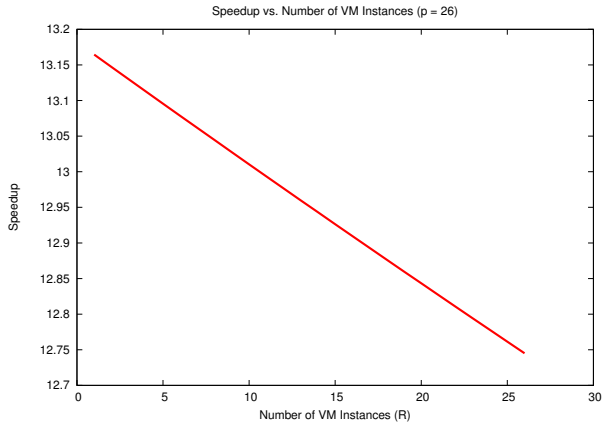


Figure 4.2: Characteristics of the valuation function: Speedup vs. number of VM instances ($p_j = 26$)

reaches about 25, which is the optimal number of processors. In Figure 4.2, we plot the speedup vs. the number of VM instances (R_j), where the number of processors is kept constant at $p_j = 26$. The speedup decreases as the number of VM instances increases.

The speedup function given in Equation (4.3) serves as the basis for the valuation function and the bidding strategy algorithm that we propose.

4.2.2 Valuation Function and Algorithm

We now determine a valuation function which gives the 'value' a user receives upon completion of her application on a bundle of VM instances. The function should depend on the parameters of the application and the bundle allocated to it. We define the valuation for using a bundle for one unit of time as follows:

$$v_j = (\lambda_j S_j)(S_j/S_{opt}) \tag{4.4}$$

Here, S_j is given by Equation (4.3) and λ_j denotes how much user u_j values each unit of speedup. The valuation can be interpreted as follows. When the optimal speedup can be achieved (i.e., $S_j = S_{opt}$) within the budget of the application the valuation function is directly proportional to the speedup. When the optimal speedup cannot be achieved the valuation is proportional to the 'scaled speedup', where the scaling factor is given by $S_j/S_{opt} < 1$. That is, if the optimal speedup is not achieved the user values less the unit of speedup than in the case in which the optimal speedup is achieved.

The valuation v_j can also be interpreted as the user's maximum willingness to pay for the given bundle. This value is the bid amount that the user submits to the auctioneer (i.e., cloud provider) along with her requested bundle. If the bundle is allocated, user u_j needs to pay a price y_j determined by the auction. The user's *utility* is defined as the surplus value retained by the user after paying the price for executing her application.

$$U_j = \begin{cases} v_j - y_j & \text{if } x_j = 1 \\ 0 & \text{if } x_j = 0 \end{cases} \tag{4.5}$$

Here, x_j is the indicator variable of whether user u_j receives her requested bundle of VM instances. Obviously, the user's goal is to maximize her utility.

Now we present our proposed Efficient Bidding Strategy (EBS). As we mentioned before, an auction is run each unit of time (e.g., every hour). Therefore, a bidding strategy needs to

Algorithm 4 Efficient Bidding Strategy (EBS)

Require: ω_j, τ_j, c_j, V_j, λ_j
Ensure: $(B_j = (r_1^j, \ldots, r_m^j, v_j), T_j)$
{Phase I: Finding optimal bundle and speedup}
$p_j \leftarrow \lfloor \sqrt{\omega_j/(\theta + \delta\tau_j\omega_j)} \rfloor$
$z \leftarrow p_j$
$R_j \leftarrow 0$
for $i := m$ **downto** 1 **do** {initialize r_i^j}
  $r_i^j \leftarrow \lfloor z/w_i \rfloor$
  $z \leftarrow z \mod w_i$
  $R_j \leftarrow R_j + r_i^j$
end for
$T_j \leftarrow \omega_j/p_j + (p_j - 1)\theta + (p_j - 1)\delta\tau_j\omega_j + (R_j - 1)\delta c_j\omega_j$
$S_j \leftarrow \omega_j/T_j$
$S_{opt} \leftarrow S_j$ {save value of optimal speedup}
{Phase II: Finding optimal valuation and adjust if necessary}
$v_j \leftarrow (\lambda_j S_j)$
while $v_j T_j > V_j$ **and** $p_j > 0$ **do**
  $h \leftarrow 0$
  for $i := m$ **downto** 1 **do** {increase number of VMs}
    if $h = 0$ **and** $r_i^j > 0$ **then**
      $r_i^j \leftarrow r_i^j - 1$
      $R_j \leftarrow R_j - 1$
      $h \leftarrow i$
      $z \leftarrow w_i$
    else if $h > 0$ **then**
      $r_i^j \leftarrow r_i^j + \lfloor z/w_i \rfloor$
      $R_j \leftarrow R_j + \lfloor z/w_i \rfloor$
      $z \leftarrow z \mod w_i$
    end if
  end for
  if $h = 0$ **then** {maximum possible R_j reached}
    $p_j \leftarrow p_j - 1$
    $z \leftarrow p_j$
    $R_j \leftarrow 0$
    for $i := m$ **downto** 1 **do** {initialize r_i^j for new p_j}
      $r_i^j \leftarrow \lfloor z/w_i \rfloor$
      $z \leftarrow z \mod w_i$
      $R_j \leftarrow R_j + r_i^j$
    end for
  end if
  $T_j \leftarrow \omega_j/p_j + (p_j - 1)\theta + (p_j - 1)\delta\tau_j\omega_j + (R_j - 1)\delta c_j\omega_j$
  $S_j \leftarrow \omega_j/T_j$
  $v_j \leftarrow (\lambda_j S_j)(S_j/S_{opt})$ {scale valuation w.r.t. speedup}
end while
if $p_j = 0$ **then**
  return "Budget too low"
end if
return $(B_j = (r_1^j, \ldots, r_m^j, v_j), T_j)$

determine the bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$ and the time required to complete the application, T_j. Here r_i^j is the number of VM instances of type VM_i included in the requested bundle of u_j. The goal of the strategy is to find a bundle and valuation so that the user can obtain the best possible speedup within her budget V_j, and thus, maximize her utility. Since v_j is the value the user is willing to pay for the allocation of the bundle for one unit of time, the algorithm has to ensure that the maximum total cost for the user does not exceed her budget, i.e.,

$$v_j T_j \leq V_j \tag{4.6}$$

Our bidding strategy algorithm presented in Algorithm 4 consists of two phases. In the first phase (lines 1 to 12), it determines the optimum number of processors (i.e., the number of processors that gives the maximum speedup). First, the optimal number of processors is calculated (line 2). To compute the optimal number of processors we consider that all processors are part of the same VM, that is, we set $R_j = 1$ in Equation (4.2). We then differentiate T_j with respect to p_j and solve $\frac{dT_j}{dp_j} = 0$ for p_j as follows.

$$-{p_j}^2 + \theta + \delta\tau_j\omega_j = 0 \tag{4.7}$$

Solving the above equation gives us the optimal value of p_j as:

$$p_{opt} = \lfloor\sqrt{\omega_j/(\theta + \delta\tau_j\omega_j)}\rfloor \tag{4.8}$$

We then generate the bundle with the least number of VM instances possible for p_j processors (lines 3 to 9). Lines 10 and 11 compute the optimal execution time and the speedup with respect to the optimal values of p_j and R_j, respectively. We save the value of the optimal speedup for later use (line 12).

Note that we set $R_j = 1$ while deriving the optimal number of processors in Equation (4.8). This is because the speedup function is linearly decreasing with respect to R_j (Figure 4.2). Thus, we determine first the optimal value of p_j without considering R_j. In

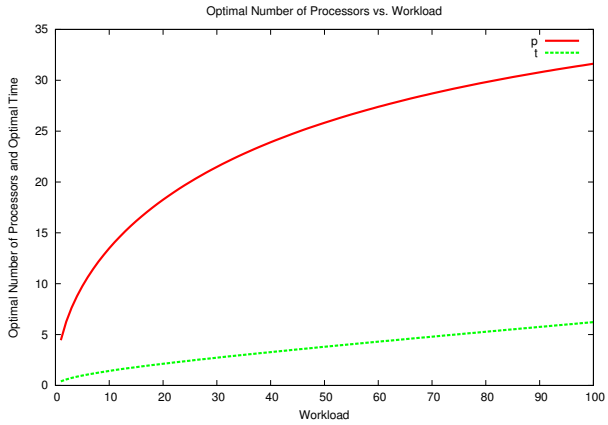


Figure 4.3: Characteristics of the valuation function: Optimal number of processors (p_j) and time (T_j) vs. workload.

Figure 4.3, we show the optimal number of processors and the optimal time as a function of the workload for the same example as the one used to plot Figure 4.2.

In the second phase, the algorithm adjusts the number of processors and/or number of VM instances if the valuation corresponding to the maximum speedup exceeds the user's budget. That is, R_j is increased if the valuation exceeds the user budget. If the valuation exceeds the budget even when $R_j = p_j$ (i.e., the bundle is composed of only one-processor VM instances), then we decrease the value of p_j and iterate until a valuation within the budget is found, or $p_j = 0$, which means that it is not possible to execute this application under the given budget V_j.

Lines 13 to 46 constitute the second phase of the algorithm. First, we calculate the optimal valuation of the bundle for allocation of one unit time, based on the optimal speedup (line 14). If this valuation cannot satisfy Equation 4.6, the while loop in lines 15-42 iterates until a feasible bundle is found or it is determined that no such bundle exists. The for loop in lines 17 to 28 selects one of the largest VM instances in the bundle and replaces it with smaller VM instances, thus increasing R_j and generating a lower value bundle. The indicator variable h initialized in line 16 captures the status of whether we can increase R_j for the current value of p_j or not. If R_j has reached its maximum, we decrease p_j, create a new bundle (lines 29-38), and compute T_j and S_j as above (lines 39-

40). For this reduced speedup, we compute v_j as before, but then scale it with the ratio of S_j and the optimal speedup (line 41). The implication of this can be understood from the following equation.

$$\begin{aligned} v_j T_j &= (\lambda_j S_j)(S_j/S_{opt})T_j = (\lambda_j \omega_j / T_j)(S_j/S_{opt})T_j \\ &= (\lambda_j \omega_j)(S_j/S_{opt}) \end{aligned} \tag{4.9}$$

From the above equation, we see that scaling the valuation makes sure that the maximum total cost of executing the application depends on the ratio of the achieved and the optimal speedup. Thus, the total cost of the application decreases and the loop in line 15 terminates when a solution is found within the budget constraint. The algorithm returns the bid and the time requirement if a feasible solution is found, otherwise it returns "Budget too low".

4.2.3 Analysis of EBS

Effectiveness. From Equation (4.3), we see that the speedup of a malleable parallel application is dependent on two variables, p_j and R_j. One way to optimize the speedup would be to solve a two variable unconstrained optimization problem. We did not consider solving the two variable optimization problem for two reasons. First, variables p_j and R_j limit each others values. Since there are certain types of VM instances available from the cloud provider, a given number of processors cannot be mapped into an arbitrary number of VM instances. Therefore, a solution from solving the two-variable optimization problem might not be practically implementable and need further adjustments. On the other hand, Figures 4.1 and 4.2 suggests that only p_j has a non-linear relationship with the speedup. Hence, finding the optimum p_j and setting the lowest possible value for R_j will give us one best estimate of the optimal speedup. If the best speedup leads to a valuation that exceeds the budget, the algorithm is capable of finding the best speedup within the budget.

Since the EBS algorithm ensures the best speedup within the budget, this maximizes the user's utility. This is because the user bids her true valuation in a combinatorial auction

that is incentive-compatible [71].

Running Time. In Algorithm 4, the while loop starting in line 15 dominates the running time. In the worst case, this loop will iterate until $p_j = 0$. In that case, the for loop in line 17 iterates p_j times for each value of p_j, where $1 \leq p_j \leq p_{opt}$. Each iteration of this loop replaces one larger VM instance with two or more smaller VM instances until the bundle contains p_j one-processor VM instances. Therefore, for one value of p_j, the loop in line 17 iterates p_j times in the worst case. The worst-case running time of the algorithm is $\sum_{i=1}^{p_{opt}} i = O({p_{opt}}^2)$.

4.3 Experimental Results

We perform simulation experiments to evaluate the proposed bidding strategy. Our goal is to answer the following questions: (i) How does the bidding strategy affect users' utilities? (ii) How does the bidding strategy affect the cloud providers' revenue? (iii) How does the number of strategic bidders participating in an auction influence other bidders' utilities?

4.3.1 Naive Bidding

We compare our proposed strategy with a 'naive strategy'. The naive bidding strategy does not consider the application's speedup function when determining the bid, but it is not completely arbitrary. A naive bidder has different values for different types of VM instances and assumes the existence of some overhead which she incorporates in her bids. The Naive-Biding (NB) algorithm is given in Algorithm 5.

The naive user u_j assigns a value ρ_i^j to each VM_i as follows. She bases the assignment on a known fixed-price vector $f_1, \ldots, f_m$ of a cloud provider where $f_i/f_{i'} = w_i/w_{i'}$, for $i = 1, \ldots, m$. The value for VM_1 is determined by multiplying f_1 by a valuation factor σ_1^j (line 1). The values of the other VMs, ρ_i^j, $i = 2, \ldots, m$, are determined by multiplying σ_1^j by the second valuation factor, σ_2^j, and by the relative values of other VM_i instances,

Algorithm 5 Naive-Bidding (NB)

Require: ω_j, V_j, σ_1^j, σ_2^j, β_1^j, β_2^j
Ensure: $(B_j = (r_1^j, \ldots, r_m^j, v_j), t_j)$
1: $\rho_1^j \leftarrow \sigma_1^j f_1$
2: **for** $i := 2$ **to** m **do** {set user's 'own' value for VM_i}
3: $\quad \rho_i^j \leftarrow \sigma_2^j \rho_1^j (w_i / w_1)$
4: **end for**
5: **repeat** {select a random VM type VM_i}
6: $\quad i \leftarrow random(1, m)$
7: $\quad$ {select r_i^j, t_j such that $r_i^j t_j > \omega_j / w_i$}
8: $\quad z \leftarrow \omega_j / w_i$
9: $\quad r_i^j \leftarrow \lceil \beta_1^j \sqrt{z} \rceil$
10: $\quad t_j \leftarrow \lceil \beta_2^j z / r_i^j \rceil$
11: $\quad v_j \leftarrow \rho_i^j r_i^j$
12: $\quad$ {reject VM_i if budget exceeded}
13: **until** $v_j t_j \leq V_j$ **or** all VM types are rejected
14: **if** $v_j t_j > V_j$ **then**
15: $\quad$ **return** "Budget too low"
16: **end if**
17: **return** $(B_j = (0, \ldots, r_i^j, \ldots, 0, v_j), t_j)$

$i = 2, \ldots, m$ (lines 2-4).

The bid is generated in lines 6 to 12. First, a VM type is selected at random (line 6). Then the user creates a bundle and determines the time required to execute her application so that her application can be completed based on her assumption about the overhead, given by overhead factors β_1^j and β_2^j. The basis of generating the bundle is that with no overhead, r_i^j instances of VM_i should execute an application a_j with workload ω_j in t_j units of time if $w_i r_i^j t_j = \omega_j$. Given w_i, one set of values of r_i^j and t_j that satisfies this condition is $r_i^j = t_j = \sqrt{z}$, where $z = \omega_j / w_i$ (line 8). The user tries to avoid requesting an excessive number of VM instances (i.e., making an attempt to reduce the overhead) by setting r_i^j as $\sqrt{z}$ times overhead factor β_1^j (line 9). t_j is computed as the value required to satisfy $r_i^j t_j = z$ times the other overhead factor β_2^j, in an attempt to allocate some extra time for completion of the job (line 10). The valuation is simply the product of r_i^j and ρ_i^j (line 11). Lines 6 to 12 are enclosed in a repeat-until loop between lines 5 to 14 that changes the VM selection when the valuation-time product exceeds the budget. The algorithm returns

"Budget too low" if no selection of VM type can satisfy the budget constraint.

4.3.2 Simulation Parameters

We perform two sets of simulation experiments. In the first set of experiments, each user takes part in two separate auctions. In the first auction, all users participate by generating their bids using the EBS algorithm while in the second auction, they all use the NB algorithm. In the following, the users that bid using EBS are called *strategic users*, while those bidding using NB are called *naive users*. A user bids in both auctions simultaneously until their application is completed or its deadline is exceeded. The parameters characterizing the application of a user are kept the same for both auctions. This set of experiments allows us to investigate the overall effect of the bidding strategies on the system.

In the second set of experiments, we run an auction with mixed user population. In each experiment, we set a distribution of naive and strategic users. Users are generated according to this distribution and all users take part in a single auction. These experiments allow us to investigate how effective our bidding strategy is on helping the users achieve better utilities. They also allow us to examine the effect of strategic bidders on the naive bidders and vice versa.

Each simulation experiment generates $N = 50,000$ users on a span of $D = 5, 10,$ or 15 simulated days. We run 24 auctions each day, one per hour. Users are generated each hour, at an average rate of $N/(24D)$ plus a randomly chosen deviation between $[-20\%, 20\%]$. The workload and the budget of a user are randomly selected from $[\omega^{min}, \omega^{max}]$ and $[V^{min}, V^{max}]$, respectively. Similarly, the parameters τ_j, c_j, λ_j, σ_1^j, σ_2^j, β_1^j, and β_2^j are randomly chosen from their respective minimum and maximum ranges presented in Table 4.1. The bidding algorithms generate the bundles of VMs, their valuations, and the time required for application completion. The deadline of a task is determined by multiplying a deadline factor d_j with the time required, T_j, thus, the deadline is $d_j T_j$. Here, d_j is chosen randomly from $[d^{min}, d^{max}]$.

We consider $m = 4$ types of VM instances in the system, each having k copies available,

Table 4.1: Simulation Parameters

Parameter	Description	Value(s)
D	Simulated days	5, 10, 15
θ	Constant setup time for an application	0.01, 0.05
δ	Communication time between VMs	0.01, 0.02
m	Types of VMs	4
k	Available VMs of each type	200, 500
$\mathbf{w}$	Relative weight of VMs	(1, 2, 4, 8)
$\mathbf{f}$	Fixed price of VMs	(.12, .24, .48, .96)
N	Maximum number of users	50,000
n	Users participating in an auction	Varies
$\omega^{min}, \omega^{max}$	Min. & Max. workload	10, 50 20, 100
V^{min}, V^{max}	Min. & Max. user's budget	5, 25 10, 50
d^{min}, d^{max}	Min. & Max. deadline factor	4, 8
τ^{min}, τ^{max}	Min. & Max. initialization overhead	1%, 5% 1%, 10%
c^{min}, c^{max}	Min. & Max. communication overhead	1%, 5% 1%, 10% 5%, 25%
$\lambda^{min}, \lambda^{max}$	Min. & Max. valuation for speedup	0.1, 0.25 0.1, 0.5
$\sigma_1^{min}, \sigma_1^{max}$	Min. & Max. valuation factor 1	0.75, 1.50 1.0, 2.0
$\sigma_2^{min}, \sigma_2^{max}$	Min. & Max. valuation factor 2	0.75, 1.50 1.0, 2.0
$\beta_1^{min}, \beta_1^{max}$ $\beta_2^{min}, \beta_2^{max}$	Min. & Max. overhead factor 1 Min. & Max. overhead factor 2	90%, 95% 105%, 125%
$\boldsymbol{\pi}$	Distribution of users	(5%, 95%), (10%, 90%), (25%, 75%), (50%, 50%)

where $k = 200, 500$. We set the weight and fixed-price vectors of VM instance types to $\mathbf{w} = (1, 2, 4, 8)$ and $\mathbf{f} = (\$0.12, \$0.24, \$0.48, \$0.96)$. These are taken from the Windows Azure's fixed-price VM allocation mechanism [41]. Finally, for the second set of experiments, the parameter $\boldsymbol{\pi}$ denotes the distribution of the two types of users. The distributions we use are $(5\%, 95\%)$, $(10\%, 90\%)$, $(25\%, 75\%)$, and $(50\%, 50\%)$, where $(5\%, 95\%)$ means that there are 5% users who bid strategically using EBS and the rest are naive bidders using NB.

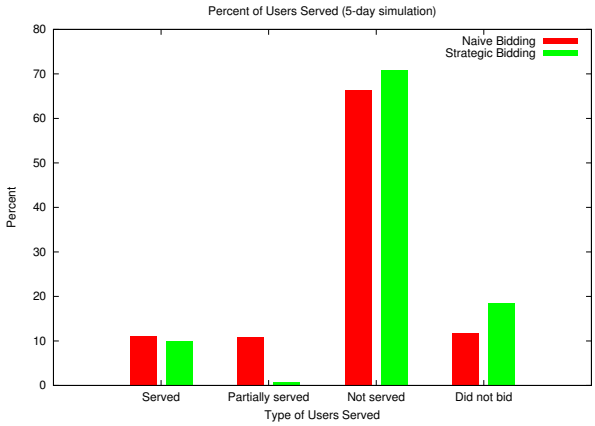


Figure 4.4: Separate auctions (five-day simulation): Users served

We also vary the system parameters θ and δ in different experiments. We list all the simulation parameters and their values in Table 4.1. By varying the parameters we perform 4,608 experiments with combinatorial auctions with separate user population and 6,144 experiments with combinatorial auctions with mixed user population.

4.3.3 Analysis of Results

First, we compare and analyze the outcomes of the experiments where each user generates a naive bid and a strategic bid for the same application and participates in two different auctions. In Figures 4.4, 4.5, and 4.6, we show the percentage of users served, the average utility, and the total revenue generated for $D = 5$ simulation days. In these figures, the users whose applications are completed at the end of the simulation are called the 'served users'. 'Partially served users' are those whose applications are only partially completed. The users whose applications did not even start fall into the category of 'not served' users. Finally, the users who did not bid because they exceeded their budget are included in the 'did not bid' group.

The utilities of the users are computed using Equation (4.5), where the valuation is determined using Equation (4.4), because this is the correct valuation of a requested bundle irrespective of the method of generating the bundle. We see that although the number of

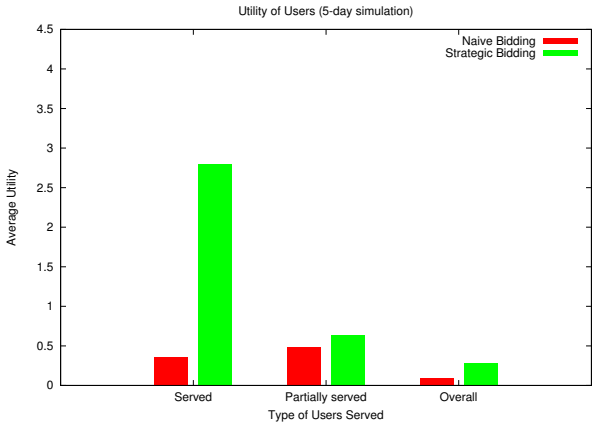


Figure 4.5: Separate auctions (five-day simulation): Average utility

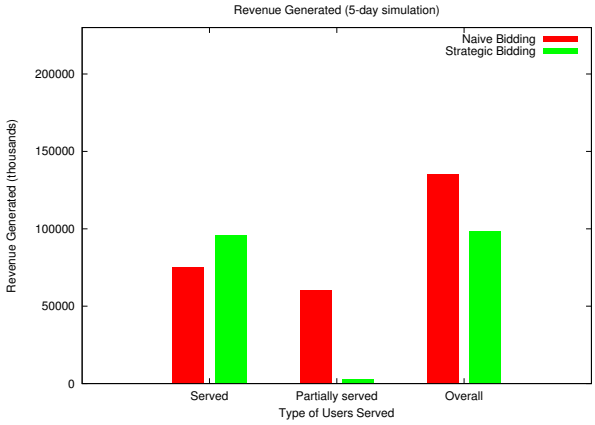


Figure 4.6: Separate auctions (five-day simulation): Total revenue

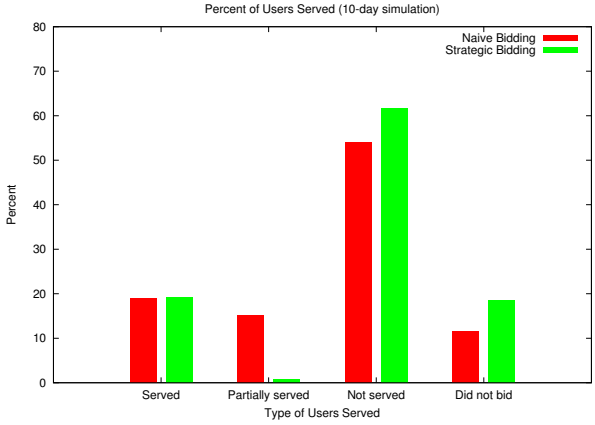


Figure 4.7: Separate auctions (ten-day simulation): Users served

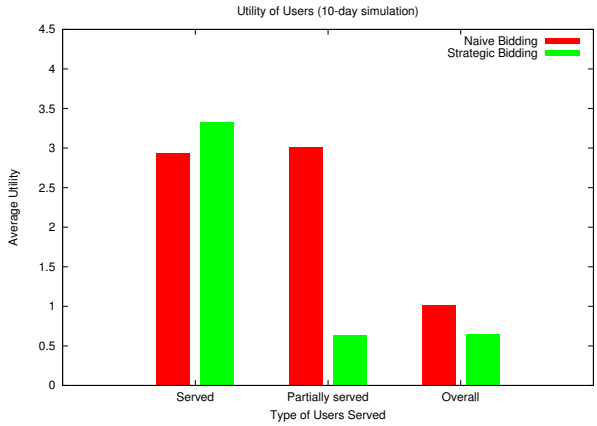


Figure 4.8: Separate auctions (ten-day simulation): Average utility

served users is slightly larger when all users bid naively, the average utility of the served users is much larger when all bidders bid strategically. The reason behind having more users served while bidding naively is that each user selects one type of VM instances at random. Therefore, there is less competition among users as compared to the other auction (with strategic bidders), where users tend to bid for the largest VM instances if the budget permits. On the other hand, since naive users are not guaranteed to bid their 'true' valuations, often a naive bidder ends up with a negative utility even when her application is completed. These factors reduce the average utility of naive users. We also see that (Figure 4.6) although the total revenue is larger when bidders are naive, a large portion of it comes from the partially served users, which is not desirable to maintain user satisfaction. The amount of revenue generated when users bid strategically consists mainly from the revenue generated by the served users.

Next, we show the results obtained by varying the overall demand on the system. We do this by expanding the simulation into ten and fifteen days, yet keeping the total number of users the same. These two sets of experiments enable us to investigate whether the performance of the EBS algorithm scales in different demand settings. These results are drawn from the experiments where each user generates one bid of each kind and participates in separate auctions. In Figures 4.7 to 4.9, we plot the percentage of served users, total

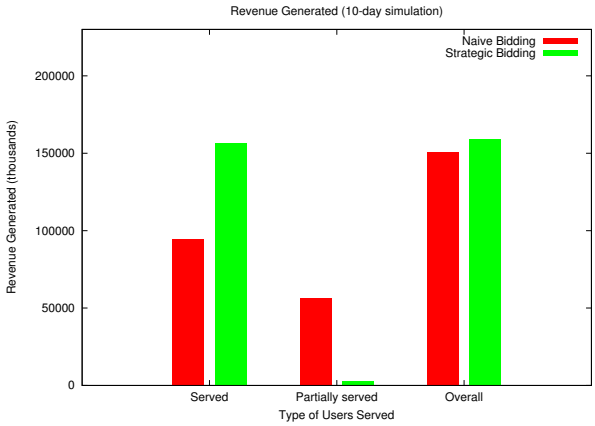


Figure 4.9: Separate auctions (ten-day simulation): Total revenue

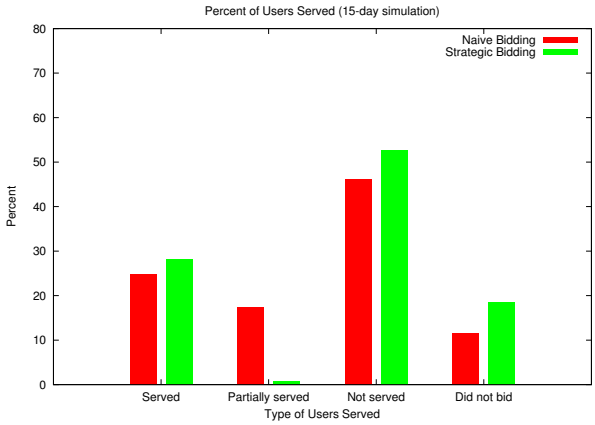


Figure 4.10: Separate auctions (fifteen-day simulation): Users served

revenue, and average utility of users where the simulations run for ten simulated days. Figures 4.10 to 4.12 show the results for the experiments that run for fifteen simulated days. Comparing Figures 4.4, 4.7, and 4.10, we see that the percentage of served users increases as the demand decreases, which is expected. We also see that the percentage of users served increases more for the EBS algorithm and in the case of fifteen-day simulation it dominates the one obtained by the NB algorithm. Although the percentage of served users is better with EBS when the demand is low, we see a different trend while comparing the average utility of users (Figures 4.5, 4.8, and 4.11). Although naive bidders that are served have comparable utility with strategic bidders when the demand is low, EBS is able

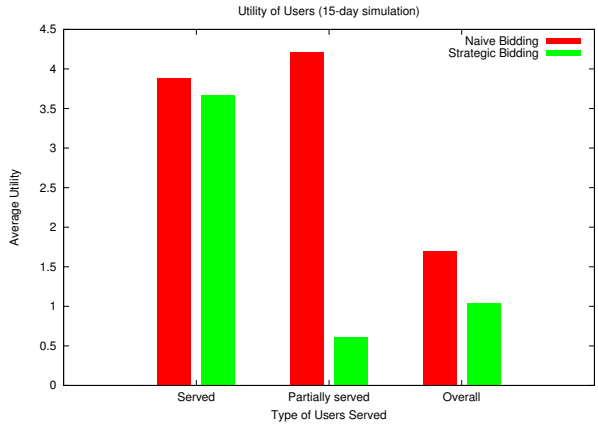


Figure 4.11: Separate auctions (fifteen-day simulation): Average utility

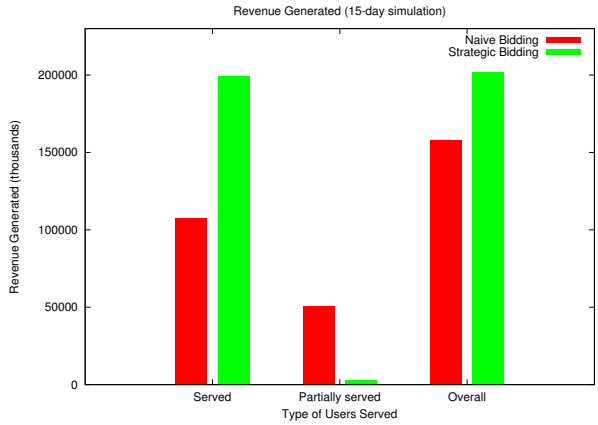


Figure 4.12: Separate auctions (fifteen-day simulation): Total revenue

to generate high utility for the users when the demand grows. However, the average utility decreases due to higher competition, which pushes the prices up. Finally, we see that the total revenue generated by the auctions increases with lower demand (as more users can be served) for both bidding algorithms. Also, when the demand is low, the bids produced by EBS generate higher revenue than the ones produced by NB.

We now present some plots to show the auction outcomes as the ranges of workload and budget vary. Figures 4.13, 4.14, and 4.15 show the percentage of served users, the average utility of served users, and the average revenue collected from the served users versus four 'scenarios' of workload and budget. The scenarios are different combination of

Figure 4.13: Users served. Here, each scenario represents a combination of $(\omega^{min}, \omega^{max}, V^{min}, V^{max})$-values. The values are: Scenario 1 $\equiv (10, 50, 5, 25)$, Scenario 2 $\equiv (10, 50, 10, 50)$, Scenario 3 $\equiv (20, 100, 5, 25)$, and Scenario 4 $\equiv (20, 100, 10, 50)$.

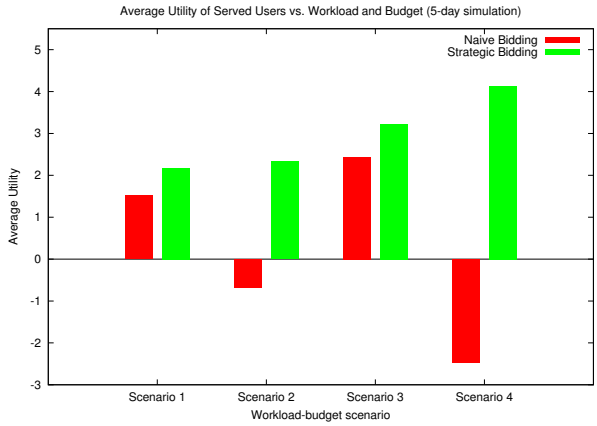


Figure 4.14: Average utility. Here, each scenario represents a combination of $(\omega^{min}, \omega^{max}, V^{min}, V^{max})$-values. The values are: Scenario 1 $\equiv (10, 50, 5, 25)$, Scenario 2 $\equiv (10, 50, 10, 50)$, Scenario 3 $\equiv (20, 100, 5, 25)$, and Scenario 4 $\equiv (20, 100, 10, 50)$.

the minimum and maximum workload and budget. For example, Scenario 1 represents the experiments where the workload varies between 10 and 50 and the budget ranges between 5 and 25. We see in Figure 4.13 that strategic bidding performs better than naive bidding in terms of the percentage of users served when the workload is higher. This illustrates that the probability of arbitrarily generating an efficient bid decreases as the workload increases. This is because the optimal number of processors and time have a non-linear relationship with the workload. Without this knowledge it is not possible to generate efficient bids

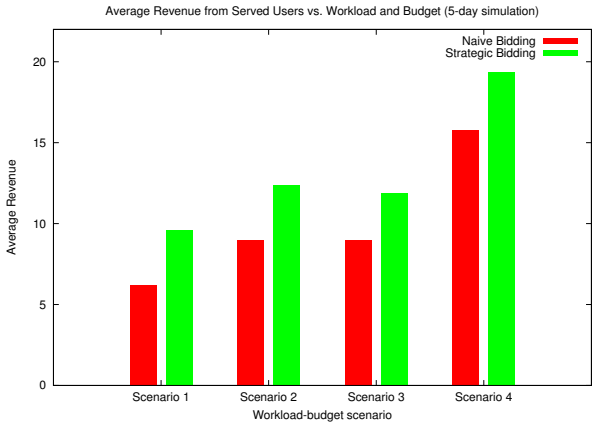
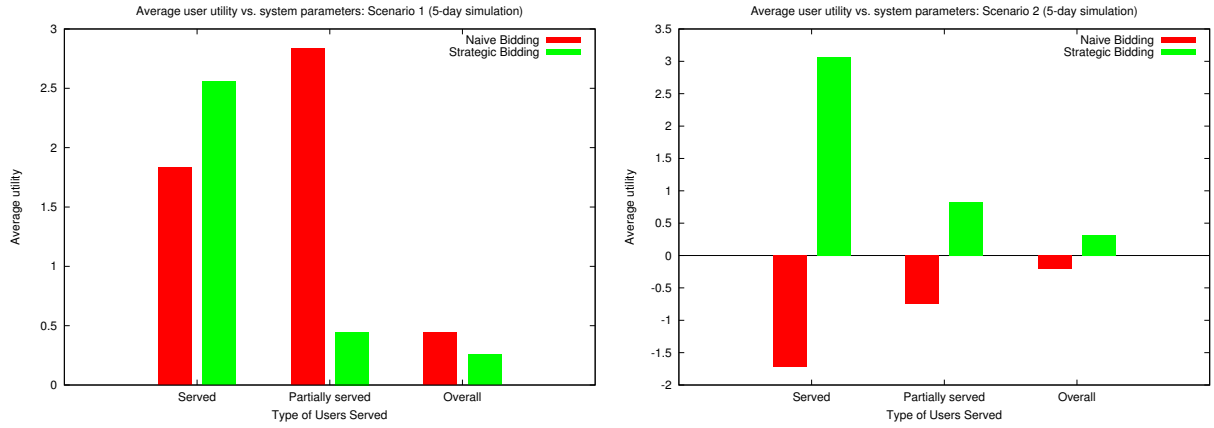


Figure 4.15: Total revenue. Here, each scenario represents a combination of $(\omega^{min}, \omega^{max}, V^{min}, V^{max})$-values. The values are: Scenario 1 $\equiv (10, 50, 5, 25)$, Scenario 2 $\equiv (10, 50, 10, 50)$, Scenario 3 $\equiv (20, 100, 5, 25)$, and Scenario 4 $\equiv (20, 100, 10, 50)$.

for any workload. The percentage of users served decreases in Scenarios 3 and 4 because here users have larger workloads, while the amount of resources are the same in all four experimental scenarios. Also, Figures 4.14 and 4.15 show that when the workload and the budget increases, the strategic bidding can yield more utility for the users and more revenue for the cloud provider. Generating more utility for the users is a desirable property of a bidding strategy. Generating more revenue for the providers is a positive side-effect.

Since the user utility is the primary performance factor for a bidding strategy, we further investigate the average user utility with respect to two more sets of simulation parameters. In Figure 4.16, we plot the average utility of the users vs. the minimum and the maximum values of the system parameters. Figure 4.16a shows the results for the case where all system parameters are given the minimum values from Table 4.1, that is, $(\theta, \delta, \tau^{min}, \tau^{max}) = (0.01, 0.01, 1\%, 5\%)$. We see that the overall average utility of all users is higher with naive bidding, which is due to the large difference in utility of the partially served users. But for the users who were served, strategic bidding yields about 20% more utility on average. On the other hand, when all the above parameters take their maximum values (Figure 4.16a), naive bidding suffers a lot as the average utility becomes negative. We also observe that in both cases the utility of served users is higher than the utility of



(a) (b)

Figure 4.16: (a) Average utility of users vs two extreme combination of values for system parameters; (a) Scenario 1 ≡ all system parameters have the minimum value; (b) Scenario 2 ≡ all system parameters have the maximum value.

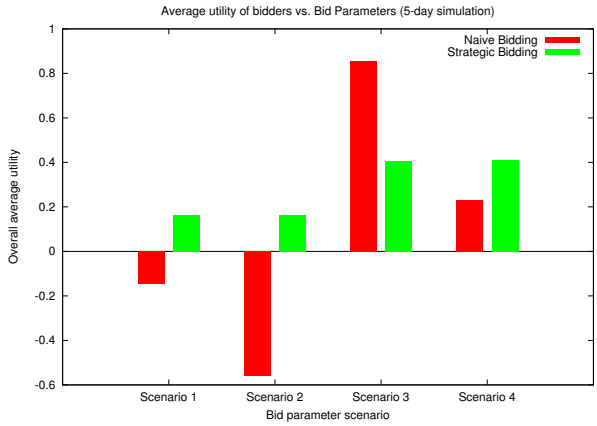


Figure 4.17: Average utility of users vs. four different scenarios of bid parameters. Here, each scenario represents a combination of $(\lambda^{min}, \lambda^{max}, \sigma_1^{min}, \sigma_1^{max}, \sigma_2^{min}, \sigma_2^{max})$-values. Scenario 1 ≡ minimum values for all parameters; Scenario 2 ≡ minimum values for λ but maximum values for both σ parameters; Scenario 3 ≡ maximum value for λ and minimum value for both σ parameters; and Scenario 4 ≡ maximum values for all parameters.

partially served users, which should be a desired property of a good bidding algorithm. The next plot, Figure 4.17, shows the average utility of users vs. different sets of values for the bid parameters (λ, σ). Here, λ^{min} and λ^{max} determine the range of the variable λ that is multiplied with speedup to determine the valuation of a bundle in EBS. On

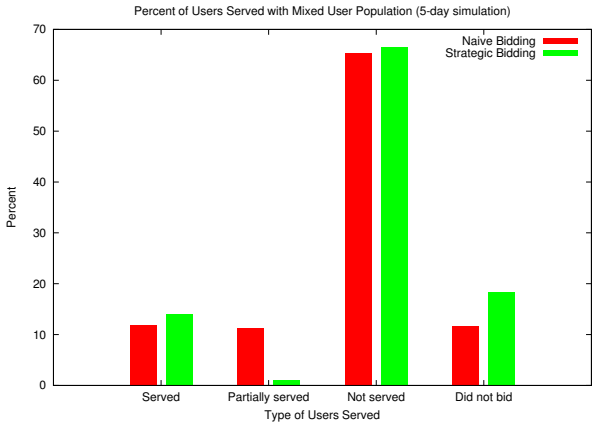


Figure 4.18: Auctions with mixed user population: Users served

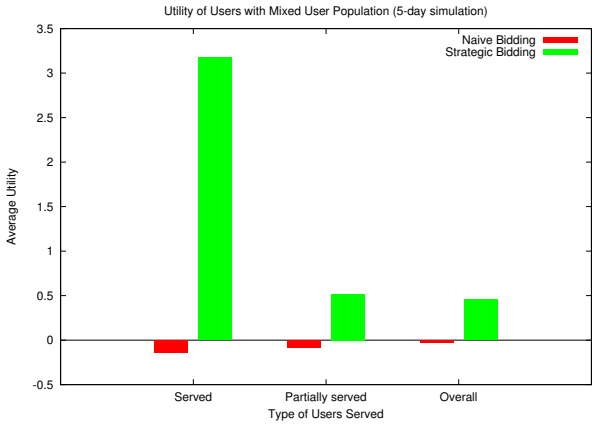


Figure 4.19: Auctions with mixed user population: Average utility

the other hand, $\sigma_1^{min}, \sigma_1^{max}, \sigma_2^{min}, \sigma_2^{max}$ determine the valuation of a bundle for the Naive-Bidding (NB) algorithm. In the plot, we show the average valuation of the users vs. the combination of values of these parameters. We see that in every case except Scenario 3 users achieve higher utility from the EBS algorithm. Scenario 3 is when λ is set to the maximum value and σ parameters are set to their minimum value. This is because in this case, EBS generates bids with much higher valuations than the NB. This raises the payment, and thus, achieves lower utility for EBS.

We now present the results from the second set of experiments, where both naive and strategic users participate in the same auction. The percentage of users served, the average

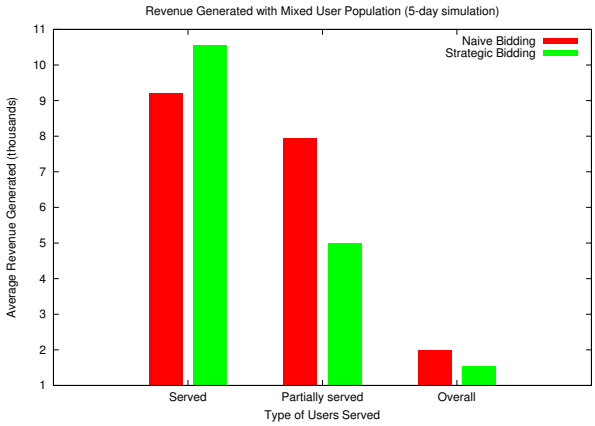


Figure 4.20: Auctions with mixed user population: Average revenue

utility and the revenue per user are shown in Figures 4.18 to 4.20. We see that in terms of served users, their utility and average revenue collected from them, the strategic bidders have better performance over the naive bidders. It is notable that in the presence of strategic bidders, the inefficiency of naive bidding is clearly seen as their average utility is negative. This is because a naive bidder is not guaranteed to bid for an optimal bundle with a true valuation. When all the participants are naive bidders, it is possible that some bidders' valuations are better than others and they receive positive utility. On the other hand, when strategic bidders are also participating, even the 'best' naive bidders are not always capable of getting the job done along with receiving a positive utility.

In the experiments with mixed user population, we also vary the percentage of users that bid strategically. In Figure 4.21, we show the percentage of strategic bidders among the served users versus the percentage of the strategic bidders in the distribution. It shows, for example, that when 5% of the users are strategic bidders, about 8% of the served bidders are strategic. This trend follows for 5%, 10%, and 25% strategic users. But when the distribution includes 50% strategic users, the percentage of served users falls below 50%. This is because of the increased competition between the strategic bidders. We also plot the average utility of served users in Figure 4.22. Here we observe that the naive bidders could gain some positive utility on average when there are only a 5% strategic users.

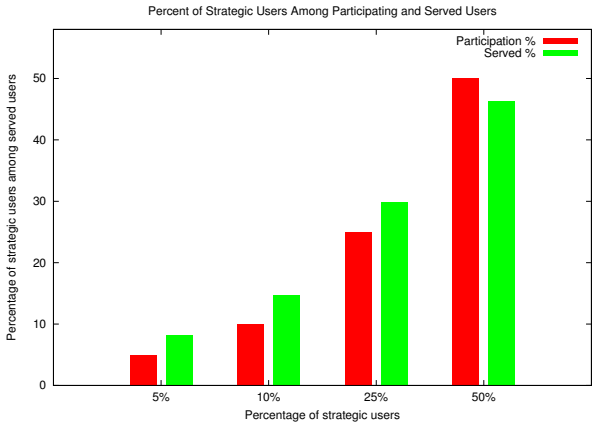


Figure 4.21: Percent of strategic users among served users vs. user distribution.

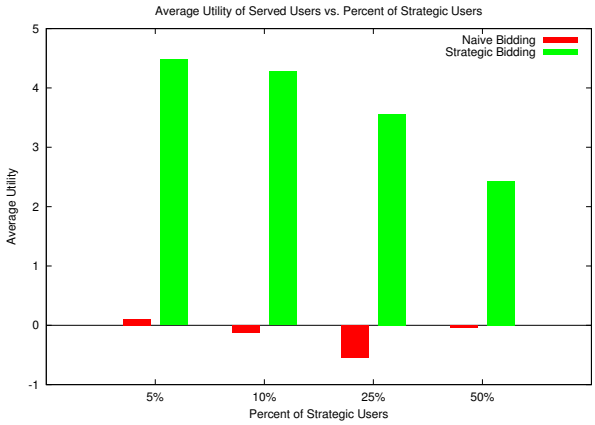


Figure 4.22: Average utility of served users vs. user distribution.

In all other cases, the naive bidders receive negative average utility. This shows how the naive bidders' utility is affected by the presence of strategic bidders. The average utility of strategic bidders decreases with more strategic users participating since the competition increases. We also examine the average revenue generated from served users when the percentage of strategic users varies in Figure 4.23. We see that as the percentage of served users increases, the average revenue generated from them increases and the average revenue from the naive users decreases. This result shows that the presence of strategic users also benefits the cloud provider.

We can conclude that by using the EBS Algorithm the users are able to generate efficient

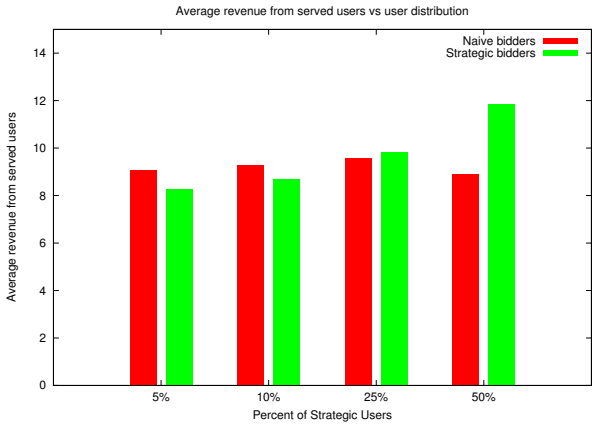


Figure 4.23: Average revenue generated from served users vs. user distribution.

bids, and thus, ensure good completion times and higher utility for their applications. This also helps the cloud provider generate a fair amount of revenue while maintaining user satisfaction.

4.4 Summary

We investigated the problem of generating efficient bids in combinatorial auction-based VM allocation mechanisms in clouds. We designed an efficient bidding algorithm and performed extensive simulation experiments to investigate its properties. We believe that this research will encourage users and cloud providers to adopt combinatorial auction-based mechanisms for allocating their VM instances.

CHAPTER 5: COMBINATORIAL AUCTION-BASED DYNAMIC VM PROVISIONING AND ALLOCATION IN CLOUDS

5.1 Introduction

Cloud computing systems provide the next computing infrastructure enabling users to provision remote resources for their computational needs, eliminating the upfront costs of setting up their own systems. Clouds give users the illusion of an infinite computing resource available on demand and allow them to acquire and pay for resources on a short term basis. Examples of cloud computing systems include both commercial (e.g., Microsoft Azure [42], Amazon EC2 [4]) and open source ones (e.g., Eucalyptus [45]). The usage model of cloud computing involves virtualization of computing resources. The cloud providers provision their resources into different types of virtual machine (VM) instances. These instances are then 'sold' to the users for specific periods of time. However, the fixed price-based resource allocation and trading mechanisms currently in use in cloud computing systems do not provide an efficient allocation of resources and do not maximize the revenue of the cloud providers. A better alternative would be to use combinatorial auction-based resource allocation mechanisms. This argument is supported by the economic theory; when the auction costs are low, as is the case in the context of cloud computing, auctions are especially efficient over the fixed-price markets since products are matched to customers having the highest valuation [67]. In particular, since each user pays a fixed price for an item, the fixed-price mechanisms cannot guarantee that the user who values an item the most gets

it. An auction-based mechanism can achieve the economic efficiency because it allocates items based on the perceived values of the users. The nature of allocation requests for cloud resources suggests that a combinatorial auction-based mechanism is best suited for the VM allocation problem in clouds. However, we have to overcome certain challenges while using combinatorial auction-based mechanisms for VM provisioning and allocation in clouds. The winner determination in a combinatorial auction is an NP-complete problem [44], therefore we need approximation algorithms to solve it.

In the previous chapter, we presented two combinatorial auction-based approximation mechanisms for VM instance allocation. Although these mechanisms are able to increase the allocation efficiency of VM instances and also increase the cloud provider's revenue, they assume static provisioning of VM instances. That is, they require that the VM instances are already provisioned and would not change. Static provisioning leads to inefficiencies due to under-utilization of resources if it cannot accurately predict the user demand. Since a regular auction computes the price of the items based on user demands, a very low demand may require the auctioneer to set a reserve price to prevent losses.

In this work, we address the VM provisioning and allocation problem by designing a combinatorial auction-based mechanism that produces an efficient allocation of resources and high profits for the cloud provider. The mechanism extends the CA-GREEDY mechanism we proposed in the previous chapter to include dynamic configuration of VM instances and reserve prices. The proposed mechanism, called CA-PROVISION, treats the set of available computing resource as 'liquid' resources that can be configured into different numbers and types of VM instances depending on the requests of the users. The mechanism determines the allocation based on the users' valuations until all resources are allocated. It involves a reserve price determined by the operating cost of the resources. The reserve price ensures that a user pays a minimum amount to the cloud provider so that the provider does not suffer any losses from the VM provisioning and allocation.

As a reminder, the results shown in Chapter 3 shows that the VM allocation problem can be best solved by combinatorial auction-based mechanisms. In that work, our focus

was to evaluate the combinatorial auctions against fixed-price mechanisms in solving the VM allocation problem with static provisioning in clouds. In this research, we design a combinatorial auction-based mechanism that dynamically provisions and allocates VM instances.

5.1.1 Our Contribution

We formulate the dynamic VM provisioning and allocation problem and provide a combinatorial auction-based mechanism to solve it. Our mechanism ensures high profits for the cloud provider, as well as high utilization of resources. We show that the mechanism is truthful, that is, it guarantees that a participating user maximizes its utility only by bidding its true valuation for the bundle of VMs. We evaluate our mechanism by performing extensive simulation experiments using traces of real workloads from the Parallel Workloads Archive [23]. We analyze the cost and benefit of employing this new mechanism and provide implementation guidelines.

5.1.2 Organization

The rest of this chapter is organized as follows. In Section 5.2, we formulate the problem of dynamic VM provisioning and allocation in clouds. In Section 5.3, we present our proposed mechanism for solving the VM provisioning and allocation problem and characterize its theoretical properties. In Section 5.4, we perform extensive simulations using real workload traces to investigate the properties of our proposed mechanism. In Section 5.5, we summarize our work presented in this chapter.

5.2 Dynamic VM Provisioning and Allocation Problem

Virtualization technology allows the cloud computing providers to configure computational resources into virtually any combination of different types of VMs. Hence, it is possible to determine the best combination of VM instances through a combinatorial auction and then dynamically provision them. This will ensure that the number of VM instances of different types are determined based on the market demand and then allocated efficiently to the users. We formulate the *Dynamic VM Provisioning and Allocation Problem (DVMPA)* as follows.

A cloud provider offers computing services to users through m different types of VM instances, $VM_1, \ldots, VM_m$. The computing power of a VM instance of type VM_i, $i = 1, \ldots, m$ is w_i, where $w_1 = 1$ and $w_1 < w_2 < \ldots < w_m$. We denote by $\mathbf{w} = (w_1, w_2, \ldots, w_m)$ the vector of computing powers of the m types of VM instances. In the rest of this chapter we will refer to this vector as the 'weight vector'. As an example of how we use this vector, let us consider a cloud provider offering three types of VM instances: VM_1, consisting of one 2 GHz processor, 4 GB memory, and 500 GB storage; VM_2, consisting of two 2 GHz processors, 8 GB memory, and 1 TB storage; and VM_3, consisting of four 2 GHz processors, 16 GB memory, and 2 TB storage. The weight vector characterizing the three types of VM instances is thus $\mathbf{w} = (1, 2, 4)$. We assume that the cloud provider has enough resources to create a maximum of M VM instances of the least powerful type, VM_1. The cloud provider can provision the VM instances in several ways according to the specified types given by $VM_1, \ldots, VM_m$. Let's denote by k_i the number of VM_i instances provisioned by the cloud provider. The provider can provision any combination of instances given by the vector $(k_1, k_2, \ldots, k_m)$ as long as

$$\sum_{i=1}^{m} w_i k_i \leq M. \tag{5.1}$$

We consider n users $u_1, \ldots, u_n$ who request computing resources from the cloud provider specified as bundles of VM instances. A user u_j requests VM instances by submitting a bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$ to the cloud provider, where r_i^j is the number of instances of type VM_i requested and v_j is the price user u_j is willing to pay to use the requested bundle of VMs for a unit of time. An example of a bid submitted by a user to a cloud provider that offers three types of VMs can be $B_j = (2, 1, 4, 10)$. This means that the user is bidding ten units of currency for using two instances of type VM_1, one instance of type VM_2, and four instances of type VM_3 for one unit of time. The provider runs a mechanism, in our case an auction, periodically (e.g., once an hour) to provision and allocate the VM instances such that its profit is maximized. In order to define the profit obtained by the cloud provider we need to introduce additional notation. Let's denote by p_j the amount paid by user u_j for using her requested bundle of VMs. Note that depending on the pricing and allocation mechanism used by the cloud provider p_j and v_j can have different values, usually $p_j < v_j$.

Let us assume that the time interval between two consecutive auctions is one unit of time. Let c_R and c_I be the costs associated with running, respectively idling a VM_1 instance for one unit of time. Obviously, $c_R > c_I$. The cloud provider's cost of running all available resources (i.e., all M VM_1 instances) is $M \cdot c_R$ while the cost of keeping all the available resources idle is $M \cdot c_I$. We denote by $x = (x_1, x_2, \ldots, x_n)$ the allocation vector, where $x_j = 1$ if the bundle $(r_1^j, \ldots, r_m^j)$ requested by user u_j is allocated to her, and $x_j = 0$, otherwise. Given a particular allocation vector and payments, the cloud provider's profit is given by

$$\Pi = \sum_{j=1}^{n} x_j p_j - c_R \sum_{j=1}^{n} x_j s_j - c_I \left(M - \sum_{j=1}^{n} x_j s_j \right) \tag{5.2}$$

where $s_j = \sum_{i=1}^{m} w_i r_i^j$ is the amount of 'unit' computing resources requested by user u_j. The 'unit' computing resource is equivalent to one VM instance of type VM_1 (i.e., the least powerful instance offered). The first term of the equation gives the revenue, the second term gives the running cost of the VM instances that are allocated to the users, and the third term gives the cost of keeping the remaining resources idle.

The problem of *Dynamic VM Provisioning and Allocation (DVMPA)* in clouds is defined as follows

$$\max \Pi \tag{5.3}$$

subject to:

$$\sum_{j=1}^{n} s_j \leq M \tag{5.4}$$

$$x_j \in \{0,1\}, \quad j = 1, \ldots, n \tag{5.5}$$

$$0 \leq p_j \leq v_j, \quad j = 1, \ldots, n \tag{5.6}$$

The solution to this problem consists of allocation x_j and price p_j for each user u_j who requested the bundle $(r_1^j, \ldots, r_m^j)$, $j = 1, \ldots, n$. The allocation will determine the number of VMs of each type that needs to be provisioned as follows. We compute $k_i = \sum_{j=1}^{n} x_j r_i^j$, for each type VM_i and provision k_i VM instances of type VM_i.

Current cloud service providers use a fixed-price mechanism to allocate the VM instances and rely on statistical data to provision the VMs in a static manner. In Chapter 3, we have shown that combinatorial auction-based mechanisms can efficiently allocate VM instances in clouds generating higher revenue than the currently used fixed-price mechanisms. However, the combinatorial auction-based mechanisms we explored in Chapter 3 require that the VMs are provisioned in advance, that is, they require static provisioning. We argue that the overall performance of the system can be increased by carefully selecting the set of VM instances in a dynamic fashion which reflects the market demand at the time when an auction is executed. In the next section, we propose a combinatorial auction-based mechanism that solves the DVMPA problem by determining the allocation, pricing, and the best configuration of VMs that need to be provisioned by the cloud provider in order to obtain higher profits. Since very little is know about profit maximizing combinatorial auctions [44], we cannot provide theoretical guarantees that our auction-based mechanism maximizes the profit. The only guarantee we can provide is that the mechanism maximizes

the sum of the users' valuations. In designing our mechanism we also use reserve prices which are known to increase the revenue of the auctioneer, in our case, the revenue of the cloud provider.

5.3 Combinatorial Auction-Based Dynamic VM Provisioning and Allocation Mechanism

We present a combinatorial auction-based mechanism, called CA-PROVISION, that computes an approximate solution to the DVMPA problem. That is, it determines the prices the winning users have to pay, and the set of VM instances that need to be provisioned to meet the winning users' demand. The mechanism also ensures that the maximum possible number of resources are allocated and no VM instance is allocated for less than the reserve price. The design of the mechanism is based on the ideas presented in [35].

CA-PROVISION uses a reserve price to guarantee that users pay at least a given amount determined by the cloud provider. Thus, the cloud provider needs to set the reserve price, denoted by v_{res}, to a value which depends on its costs associated with running the VMs. To do that we observe that the reserve price should be the break-even point between c_R and c_I, which is given by $c_R - c_I$. This is because if a unit resource is not allocated, it incurs a loss of c_I. Again, if this resource is allocated for a price $c_R - c_I$, the loss is $c_R - (c_R - c_I) = c_I$. In other words, the minimum price a user has to pay for using the least powerful VM for a unit of time is equal to the difference between the cost of running and the cost of keeping the resource idle. An auction with reserve price v_{res} can be modeled by an auction without reserve price in which we artificially introduce a dummy bidder u_0 having as its valuation the reserve price, i.e., $v_0 = v_{res}$. The dummy user u_0 bids $B_0 = (1, 0, \ldots, 0, v_{res})$, i.e., $r_1^0 = 1$, $r_i^0 = 0$ for all $i = 2, \ldots, m$, and $v_0 = v_{res}$. CA-PROVISION uses the density of the bids to determine the allocation. User u_j's *bid density* is $d_j = v_j/s_j$, where $s_j = \sum_{i=1}^{m} w_i r_i^j$, $j = 0, \ldots, n$. The bid density is a measure of how much a user bids per unit of allocation. In

Algorithm 6 CA-PROVISION Mechanism

Require: M; m; $w_j : j = 1, \ldots, n$; c_R; c_I;
Ensure: W; $p_j : j = 1, \ldots, n$; $k_i : i = 1, \ldots, m$;
1: {*Phase 1: Collect bids*}
2: **for** $j = 1, \ldots, n$ **do**
3: collect bid $B_j = (r_1^j, \ldots, r_m^j, v_j)$ from user u_j
4: **end for**
5: {*Phase 2: Winner determination and provisioning*}
6: $W \leftarrow \emptyset$ {set of winners}
7: $v_{res} \leftarrow c_R - c_I$
8: add dummy user u_0 with bid
$B_0 = (1, 0, 0, \ldots, 0, v_{res})$
9: **for** $j = 0, \ldots, n$ **do**
10: $s_j \leftarrow \sum_{i=1}^{m} r_i^j w_i$
11: $d_j \leftarrow v_j / s_j$ {'bid density'}
12: **end for**
13: re-order users $u_1, \ldots, u_n$ such that
$d_1 \geq d_2 \geq \ldots \geq d_n$
14: let l be the index such that
$d_j \geq d_0$ if $j \leq l$, and
$d_j < d_0$ otherwise
15: discard users $u_{l+1}, \ldots, u_n$
16: rename user u_0 as u_{l+1}
17: set $n \leftarrow l + 1$
18: $R \leftarrow M$
19: **for** $j = 1, \ldots, n-1$ **do** {leave out dummy user}
20: **if** $s_j \leq R$ **then**
21: $W \leftarrow W \cup u_j$
22: $R \leftarrow R - s_j$
23: **end if**
24: **end for**
25: **for** $i = 1, \ldots, m$ **do** {determine VM configuration}
26: $k_i \leftarrow \sum_{j:u_j \in W} r_i^j$
27: **end for**
28: {*Phase 3: Payment*}
29: **for all** $u_j \in W$ **do**
30: $W'_j \leftarrow \{u_l : u_l \notin W \wedge (v_j = 0 \Rightarrow u_l \in W)\}$
31: $l \leftarrow$ lowest index in W'_j
32: $p_j \leftarrow d_l s_j$
33: **end for**
34: **for all** $u_j \notin W$ **do**
35: $p_j \leftarrow 0$
36: **end for**
37: **return** $(W; p_j : j = 1, \ldots, n; k_i : i = 1, \ldots, m)$

our case the unit of allocation corresponds to one VM instance of type VM_1. To guarantee that the users are paying at least the reserve price, the mechanism will discard all users for

which $d_j < d_o$.

CA-PROVISION is given in Algorithm 6. The mechanism requires some information from the system such as the total amount of computing resources M expressed as the total number of VMs of type VM_1 that can be provisioned by the cloud provider. The mechanism also requires as input the number of available VM types, m, and their weight vector $\mathbf{w}$. It also needs to know c_R, the cost of running a VM instance of type VM_1, and c_I, the cost of keeping idle a VM instance of type VM_1.

The mechanism works in three phases. In Phase 1, it collects the users' bids B_j (lines 1 to 4). In Phase 2, the mechanism determines the winning bidders and the VM configuration that needs to be provisioned by the cloud provider as follows. It adds a dummy user u_0 with a bid that contains only one instance of VM_1 and has a valuation of $v_{res} = c_R - c_I$ (line 8). This dummy user is only used to model the auction with reserve price and will not receive any allocation. It then computes the bundle size s_j and bid density d_j of all users (lines 9 to 12). Then, all users except the dummy user are ordered in decreasing order of their bid densities and all users u_j with $d_j < d_0$ are discarded (lines 13 to 15). The dummy user u_0 is then moved to the end of the list of the remaining users since it has the lowest density in the current set of users. The mechanism reassigns n to be the total number of users under consideration, including the dummy user (lines 16 and 17).

Next, the mechanism determines the winning users in a greedy fashion. It allocates the requested bundles to users in decreasing order of their bid density, as long as there are resources available (lines 18 to 24). However, the dummy user is not considered for allocation. Once the winners are determined, the mechanism determines the VM configuration that needs to be provisioned by aggregating the bundles requested by the winning users (lines 25 to 27).

In Phase 3, the mechanism determines the payment for all users. For each winning bidder u_j the mechanism finds the set of losing bidders W'_j who would otherwise win if $v_j = 0$, i.e., when user u_j is not participating (line 30). From this set, user u_l with the highest bid density is selected. This is determined by taking the lowest indexed user from

set W'_j, since the set of users is already sorted in non-decreasing order of users' bid densities (line 31). User u_j's payment is then calculated by multiplying her bundle size s_j with the bid density d_l of u_l. All losing bidders pay zero. This type of payment is known in the mechanism design literature as the *critical payment* [35]. The reason we choose this type of payment is that it is a necessary condition for obtaining a *truthful mechanism*, (i.e., a mechanism that provides incentives to the users to bid their true valuations for the requested bundles). In the next subsection, we show that our proposed mechanism is truthful.

5.3.1 Properties of CA-PROVISION

We now investigate the properties of the proposed mechanism. An important property of a mechanism is *incentive compatibility*, which is also called *truthfulness*. This is important because the mechanism computes the allocation and payment based on the information reported by the users (i.e., bids), which is private information. A rational user may manipulate the mechanism by bidding false valuations if it benefits her to do so. The challenge of designing a mechanism, therefore, involves designing the winner determination and payment functions that give the users incentives to bid truthfully. This is very important since the users participating in a truthful allocation mechanism do not have to employ sophisticated bidding strategies to maximize their utilities. They just need to bid their true valuation for the bundle of VMs.

In the following, we denote by $B = (B_1, \ldots, B_n)$, the vector representing the bids of all users and, by $B_{-j} = (B_1, \ldots, B_{j-1}, B_{j+1}, \ldots, B_n)$ the vector of all user's bids except the bid B_j of user u_j. Hence, B can also be represented as $B = (B_j, B_{-j})$. We also assume that $B_j = (r_1^j, \ldots, r_m^j, v_j)$ is the 'true bid' of the user, i.e., the user requires the bundle $(r_1^j, \ldots, r_m^j)$ and she values it at v_j. We denote by $\hat{B}_j = (\hat{r}_1^j, \ldots, \hat{r}_m^j, \hat{v}_j)$, the bid the user submits to the mechanism, which may or may not be the same as B_j. We denote by $\hat{B} = (\hat{B}_1, \ldots, \hat{B}_n)$ the vector of all user's bids reported to the mechanism.

Here we also abuse the notations for the set of winners W and the payments $p_1, \ldots, p_n$. We will use them as the winner determination function $W(.)$ and the payment functions

$p_1(.), \ldots, p_n(.)$. $W(\hat{B})$ computes the set of winners from the bid vector $\hat{B}$ and $p_j(\hat{B})$ computes the payment for user u_j from $\hat{B}$. We express the fact that user u_j values her requested bundle at v_j by the *valuation function*

$$V_j(W(\hat{B}), B_j) = \begin{cases} v_j & \text{if } u_j \in W(\hat{B}) \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

That is, user u_j obtains a valuation of v_j if her requested bundle is allocated and a valuation of 0, otherwise.

The *utility* user u_j receives by obtaining the requested bundle is the difference between her valuation V_j and payment p_j as follows

$$U_j(W(\hat{B}), B_j) = V_j(W(\hat{B}), B_j) - p_j(\hat{B}). \tag{5.8}$$

We assume that the users are rational, that is, their goals are to maximize their utilities. A truthful mechanism guarantees that a user maximizes her utility only by bidding her true valuation for the bundle. In the following we define the concept of truthful mechanism.

Definition 1 (Truthful mechanism)**.** *A mechanism defined by the winner determination function* $W(.)$ *and payment functions* $p_1(.), \ldots, p_n(.)$ *is* truthful *if for all* u_j*,* $\hat{B}_j$*, and* $\hat{B}_{-j}$*,*

$$U_j(W(B_j, \hat{B}_{-j}), B_j) \geq U_j(W(\hat{B}_j, \hat{B}_{-j}), B_j) \tag{5.9}$$

That is, a user participating in a truthful mechanism maximizes her utility only by bidding her true valuation for the bundle regardless of the other users' bids.

Truthfulness was well investigated and characterized in the mechanism design literature [44]. One such useful characterization gives the conditions under which a mechanism is truthful. Stated informally, a mechanism is truthful if the allocation function is *monotone* and the payments are the *critical payments* [43]. We define these two properties in the context of CA-PROVISION below.

Definition 2 (Monotonicity)**.** *An allocation function $W(.)$ is* monotone *if for every user u_j and every $\hat{B}_{-j}$, $B_j = (r^j_1, \ldots, r^j_m, v_j)$ is a winning bid, then every $B'_j = (r'^j_1, \ldots, r'^j_m, v'_j)$ with $s'_j \leq s_j$ and $v'_j \geq v_j$ is also a winning bid. Here $s'_j = \sum_{i=1}^{m} w_i r'^j_i$ and $s_j = \sum_{i=1}^{m} w_i r^j_i$.*

In other words an allocation function is monotone if a winning user also wins if she bids a higher valuation for a smaller size bundle.

Definition 3 (Critical value)**.** *The critical value v^c_j for user $u_j \in W(\hat{B})$ is defined as the unique value such that $B_j = (\hat{r}^j_1, \ldots, \hat{r}^j_m, v_j)$ is a winning bid for any $v_j \geq v^c_j$ and a loosing bid for any $v_j \leq v^c_j$.*

In other words, the critical value is the minimum valuation a user must declare in order to obtain her requested bundle.

Next, we present two lemmas and one theorem to prove that CA-PROVISION is truthful.

Lemma 1. *CA-PROVISION implements a monotone allocation function.*

Proof. CA-PROVISION allocates resources to users in non-increasing order of $d_j = v_j/s_j$, where s_j is the sum of the weights of VMs in the requested bundle. Hence, a bid with higher v_j and lower s_j is preferable to the mechanism. Assume user u_j gets the allocation by bidding $B_j = (r^j_1, \ldots, r^j_m, v_j)$. If she changes her bid to $\hat{B}_j = (r^j_1, \ldots, r^j_m, \hat{v}_j)$ where $\hat{v}_j \geq v_j$, she stays at least at the same rank in the ordered list. Since she is requesting the same resource, this implies that her bid is a winning bid. On the other hand, if user u_j bids $\hat{B} = (\hat{r}^j_1, \ldots, \hat{r}^j_m, v_j)$ where $\hat{s}_j = \sum_{i=1}^{m} w_i \hat{r}^j_i \leq s_j$, then d_j increases and user u_j stays at least at the same rank in the greedy order of users (Algorithm 1, line 13). Since she is requesting fewer resources, her bid $\hat{B}_j$ is a winning bid. By Definition 2, CA-PROVISION implements a monotone allocation function. □

Lemma 2. *CA-PROVISION charges the winning users their critical payments.*

Proof. To compute the payment for a winning user u_j, CA-PROVISION finds a losing user u_l who would win if user u_j would not participate. That means user u_j needs to defeat

user u_l with her bid to get her required bundle (i.e., $d_j \geq d_l$). This means that $v_j/s_j \geq d_l$, and therefore $v_j \geq d_l \cdot s_j$. CA-PROVISION charges $p_j = d_l \cdot s_j$ to user u_j (line 32 of Algorithm 1) which is the minimum valuation u_j must bid to obtain her required bundle. Therefore CA-PROVISION implements the critical value payment (Definition 3). □

Theorem 3. *CA-PROVISION is truthful.*

Proof. According to Lemma 1 and 2, CA-PROVISION implements a monotone allocation function and charges the winning users their critical payments. Following the results of Mu'alem and Nisan [43], CA-PROVISION is a truthful mechanism. The reserve prices do not affect the truthfulness of the mechanism since they are basically bids put out by the dummy user controlled by the cloud provider and truthful bidding is still a dominant strategy for the users. □

Now, we investigate the complexity of CA-PROVISION. The loops in lines 19-24 and lines 29-33 constitute the major computational load of Algorithm 6. The first loop has a worst case complexity of $O(M)$. The worst case is when all winning bidders bid for bundles containing exactly one unit of VM_1 instances. The total execution time of the loop in lines 29-33 is $O(n)$. This is because it iterates over the set of winning bidders and the search is performed on the losing bidders. Since the bidders are already sorted, the search for a critical payment for a winner u_{j+1} actually starts from the 'critical payment bidder' u_l of u_j (without loss of generality, we assume both u_j and u_{j+1} are winners in this case). Hence, the overall worst case complexity of this loop is $O(n)$, whereas the sorting in line 13 costs $O(n \log n)$. Thus, the complexity of CA-PROVISION is $O(M + n \log n)$.

5.4 Experimental Results

We perform extensive simulation experiments with real workload data to evaluate the CA-PROVISION mechanism. We compare the performance of CA-PROVISION with the performance of the CA-GREEDY mechanism that we designed in the previous chapter. In

Chapter 3, we designed CA-GREEDY as a mechanism that allocates statically-provisioned VM instances and investigated the performance of CA-GREEDY against the performance of the fixed-price VM allocation mechanism in use by current cloud providers. The mechanism showed significant improvements over the fixed-price allocation mechanism thus making it a good candidate for our current experiments. We perform a total of 264 experiments with data generated using eleven workload logs from the Parallel Workloads Archive [23] and 24 different combination of other parameters for each workload. In this section, we describe the experimental setup and discuss the experimental results.

5.4.1 Experimental Setup

The experiments consist of generating job submissions from a given workload and then running both CA-GREEDY and CA-PROVISION concurrently to allocate the jobs and provision the VMs. For setting up the experiments we have to address several issues such as workload selection, bid generation, and setting up the auction. We discuss all these issues in the following subsections.

Workload selection

To the best of our knowledge, standard cloud computing workloads were not publicly available at the time of writing this chapter. Thus, to overcome this limitation we rely on well studied and standardized workloads from The Parallel Workloads Archive [23]. This archive contains a rich collection of workloads from various grid and supercomputing sites. Out of the twenty-six real workloads available, we selected eleven logs that were recorded most recently. These logs are: 1) ANL-Intrepid-2009, from a Blue Gene/P system at Argonne National Lab; 2) DAS2-fs0-2003 - DAS-fs4-2003, from a research grid of five clusters at the Advanced School of Computing and Imaging in the Netherlands; 3) LLNL-Atlas-2006 and LLNL-Thunder-2007 from two Linux clusters (Atlas and Thunder) located at Lawrence Livermore National Lab; 4) LLNL-uBGL-2006, from a Blue Gene/L system

Table 5.1: Workload logs

Logfile	Duration	Jobs	Processors
ANL-Intrepid-2009	8 months	68,936	163,840
DAS2-fs0-2003	12 months	225,711	144
DAS2-fs1-2003	12 months	40,315	64
DAS2-fs2-2003	12 months	66,429	64
DAS2-fs3-2003	12 months	66,737	64
DAS2-fs4-2003	11 months	33,795	64
LLNL-Atlas-2006	8 months	42,725	9,216
LLNL-Thunder-2007	5 months	121,039	4,008
LLNL-uBGL-2006	7 months	112,611	2,048
LPC-EGEE-2004	9 months	234,889	140
SDSC-DS-2004	13 months	96,089	1,664

at Lawrence Livermore National Lab; 5) LPC-EGEE-2004, from a Linux cluster at The Laboratory for Corpuscular Physics, Univ. Blaise-Pascal, France; and 6) SDSC-DS-2004, from a 184-node IBM eServer pSeries 655/690 called DataStar located at the San Diego Supercomputer Center.

In Table 5.1 we provide a brief description of the workloads we use in our experiments. The table contains the name of the log file, the length of time the logs were recorded, the total number of submitted jobs, and the total number of processors available in the system. The log file name generally contains the acronym of the organization, the name of the system, and the year of its generation. From the duration column, we see that the logs were generated for long periods of time, as long as thirteen months for the SDSC log. The number of jobs submitted ranges from many thousands to more than a couple of hundred thousands, while the number of processors ranges from 64 to 163,840. These large variations in the number of processors and the number of submitted jobs make these logs very suitable for experimentation, providing us with a wide range of simulation scenarios.

The workloads are given in the Standard Workload Format (swf) described in [24]. In this format, the information corresponding to every job submitted to the system is stored as a record with eighteen fields. To generate the workload for our simulation experiments, we need the information from six fields of the log files as follows: (1) Job Number: stores

Table 5.2: Statistics of workload logs

Logfile	Duration (hours)	Jobs / hour	Avg. Runtime	Avg procs. per job
ANL-Intrepid-2009	5759	12	2.09	5063
DAS2-fs0-2003	8744	26	1.09	10
DAS2-fs1-2003	8633	5	1.23	8
DAS2-fs2-2003	8760	8	1.29	9
DAS2-fs3-2003	8712	8	1.17	5
DAS2-fs4-2003	7963	4	1.67	4
LLNL-Atlas-2006	4308	10	2.52	401
LLNL-Thunder-2007	3605	34	1.52	43
LLNL-uBGL-2006	5339	21	1.25	576
LPC-EGEE-2004	5728	41	1.80	1
SDSC-DS-2004	9387	10	2.88	62

the job's identifier; (2) Submit Time: stores the job submission time; (3) Run Time: stores the time the job needs to complete its execution. We use this as the time required to complete the job. We round this up to the nearest hour because we run hourly auctions in the experiments. (4) Number of Allocated Processors: for our purposes this represents the number of requested processors; (5) Average CPU Time Used: Average time a CPU was running. We use this field in conjunction with the preceding two parameters to determine the amount of communication and the parallel speedup of the job. (6) User ID: stores the ID of the user who submitted the job. We use this ID to place users into different classes having different bidding behaviors. We list some statistics of the workload files in Table 5.2.

The logs from the Parallel Workloads Archive [23] were collected from different heterogeneous sources and then converted into the standard format. Therefore, in some logs, some of the fields are not specified since the original files had missing information. Some records in a log file may also have fewer fields than the other records from the same file. We make corrections on these records as follows.

- If the job starting time is missing, we consider it to be equal to the previous job's start time. The logs record the jobs in order of their arrival times. Matching a missing arrival time with the previous job maintains the job order.

- If the execution time is missing, we randomly generate an execution time between one and two hours from a uniform distribution. As can be seen in Table 5.2, most of the workloads have an average runtime within this range.

- If the number of processors is missing, we generate a number between 10 and 60 randomly, from a uniform distribution. Since the average number of processors per job differs widely among the workload logs (from 1 processor/job up to 5063 processors/job), we select a distribution that has a mean (35 processors/job) approximately equal to the average of the two-digit numbers in the list (i.e., 10, 43, and 63 processors/job).

- If the average CPU time is missing we generate a random number between 50% and 100% of the total run time using the uniform distribution. This generates jobs with communication to computation ratios between 0 and 0.5.

- We assign user IDs randomly in cases in which they are not provided.

Job and bid generation

For each record in a log file we generate a job that a user needs to execute and create a bid for it. There are two important parameters associated with a job that we need to generate, the requested bundle of VMs and the associated bid. First, to generate the bundle of VM instances for a job j, we determine its *communication to computation ratio* as follows

$$\rho_j = 1 - \frac{T_j^{CPU}}{T_j^R} \tag{5.10}$$

where T_j^{CPU} is the average CPU time and T_j^R is the total run time of the job. The communication to computation ratio measures the fraction of the total runtime that is spent by the job on communication and synchronization among its processes. Based on this value, we categorize the job into one of m categories, where m is the number of VM types available. The job category specifies a 'first choice' of VM type for the job. This

works as follows. We define a factor μ that characterizes how many of the total requested VMs will be requested as 'first choice' type VM instances. A job of category i requesting P_j processors will create a bundle comprising a number of VM_i instances required to allocate μP_j processors. The rest of the processors will be requested by arbitrarily choosing other VM types. After creating the bundle, we generate the associated bid. To do that we first determine the speedup of the job as follows

$$S_j = P_j \times \frac{T_j^{CPU}}{T_j^R} \tag{5.11}$$

where P_j is the number of CPUs used, T_j^{CPU} is the average CPU time, and T_j^R is the total run time of the job. This speedup is multiplied by a 'valuation factor' to generate the bid. This valuation factor is linked to the type of user. We divide the users into five categories using their user ID, modulo five. The last parameter we set for a job is its deadline. Since there is no deadline information provided in the workload logs, we assume that the deadline is between 4 and 8 times the time required to complete the job. Hence, we set the deadline of a job to the required time multiplied by a random number between 4 and 8.

We run CA-GREEDY and CA-PROVISION mechanisms concurrently and independently considering the users who have jobs available for execution. A user (or job) participates in the auction until her job completes or it becomes certain that the job cannot finish by the deadline. A user is 'served' if her job completes its execution and 'not served' otherwise. Without loss of generality, we assume that each user is submitting only one job and we will use 'user' and 'job' interchangeably in the rest of the chapter.

Auction setup

We consider a cloud provider that offers four different types of virtual machines instances VM_1, VM_2, VM_3, and VM_4. These VM types are characterized by the weight vector $\mathbf{w} = (1, 2, 4, 8)$. From each workload file, we extract N, the total number of users and M, the total number of processors available. The number of users participating in a particular

Table 5.3: Simulation Parameters

Name	Description	Value(s)
N	Total users	From log file
M	Total CPUs	From log file
T	Simulation hours	From log file
(c_I, c_R)	Idle and running cost of unit VM	(.05, .1), (.1, .25), (.15, .5)
μ	Factor for CPUs for 'first choice' VM type	0.5, 0.75
$\mathbf{h}$	Static distribution of processors among VM types	(.25, .25, .25, .25), (.07, .13, .27, .53)
$\mathbf{f}$	Valuation factors for types of users	(.5, 1, 1.5, 2, 2.5), (1, 1.5, 2, 3, 4)
C_1, C_2, C_3	Boundaries of communication ratios	(.05, .15, .25)

auction is determined dynamically as the auction progresses. That is, n is the number of users that have been generated, not yet been served, and whose job deadlines have not been exceeded yet.

We setup few parameters to generate bundles specific to the jobs submitted by a user. The vector (C_1, C_2, C_3) determines the communication ratios used to categorize the jobs. We use $(C_1, C_2, C_3) = (0.05, 0.15, 0.25)$, as follows. A job having communication ratio below 0.05 is a job of type 1 and the majority of its needed VM instances μp_j will be requested as VM_1, where p_j is the number of processors requested by user u_j. We consider the following values for μ, 0.5 and 0.75. The rest of the bundle is arbitrarily determined using the other types of VM instances. We use the user ID field of the log file to determine the valuation range of the user. There are five classes of users submitting jobs. The class t of a user is determined by ((user ID) mod 5). The logs have real user IDs, therefore this classification virtually creates a realistic distribution of users. Each class t of users is associated with a 'valuation factor' f_t. Having determined that a user is of class t, we determine the valuation of her bundle using the speedup (as shown in the previous subsection) and the 'valuation factor' f_t from the vector $\mathbf{f}$. The vector $\mathbf{f}$ has five elements (equal to the number of classes of users), each representing the mean value of how much a

user of that class 'values' each 'unit of speedup'. In particular, a user u_j having a speedup of S_j for her job is willing to pay $f_t S_j$ on average for each hour of her requested bundle of VMs, given that u_j falls in class t. We generate a random value between 0 and $2f_t$, and then multiply it with S_j to generate valuations with a mean of $f_t S_j$. We use two sets of vectors for $\mathbf{f}$, as shown in Table 5.3.

CA-PROVISION determines by itself the configuration of the VMs that needs to be provisioned by the cloud provider, whereas CA-GREEDY assumes static VM provisioning, and thus, needs the VM configuration provisioned in advance. To generate the static provision of VMs required by CA-GREEDY we use a vector $\mathbf{h}$ as follows. We consider two instances of $\mathbf{h}$ in the simulation. The first one, $\mathbf{h} = (0.07, 0.13, 0.27, 0.53)$ ensures that, given the weight vector $\mathbf{w}$, the number of VM instances of each type is not the same. The other instance of this vector, $\mathbf{h} = (0.25, 0.25, 0.25, 0.25)$ ensures that the total number of processors are equally distributed to different types of VMs. We list all simulation parameters in Table 5.3. With all combinations of values, we perform 24 experiments with each log file, for a total of 264 experiments.

5.4.2 Analysis of Results

We investigate the performance of the two mechanisms for different workloads. Since the workloads are heterogeneous in several dimensions, we first define a metric in order to characterize the workloads, and thus, be able to establish an order among them. Then, we normalize the performance metrics of the mechanisms and compare them with respect to the workload characteristics. Finally, we try to gain more insight by comparing the allocation determined by the two mechanisms side by side.

We define a metric for comparing the workload logs as follows. Looking at the workload characteristics listed in Tables 5.1 and 5.2, we determine that the best metric to compare the workloads is the *normalized load* defined as:

$$\eta_\omega = \frac{J_\omega \times T_\omega \times P_\omega}{M_\omega}. \tag{5.12}$$

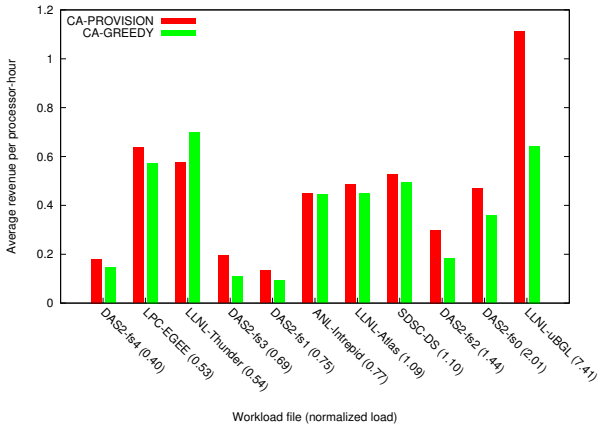


Figure 5.1: Average revenue per processor-hour by CA-PROVISION and CA-GREEDY vs. normalized load.

In the above expression, η_ω is the normalized load of workload ω, J_ω is the average number of jobs submitted per hour, T_ω is the average runtime of the jobs, P_ω is the average number of processors required per job, and M_ω is the total number of processors in the system corresponding to workload ω. The number of jobs per hour multiplied by the average processors per job determines how many processors are requested by the jobs arriving each hour. Multiplying this with the average runtime gives an estimate of the average number of processors requested by all jobs in an hour. The normalized load gives us an ordering of the set of workloads.

From each set of simulation experiments, we compute the total revenue generated, the total cost incurred, and the total profit earned by each mechanism. Since the workloads were generated for different durations of time for systems with different number of processors we scale the profit, revenue, and cost with respect to the total simulation hours and the number of processors. We define the *profit per processor-hour* as:

$$\Pi_\omega^{ph} = \frac{\Pi_\omega}{M_\omega \times L_\omega} \tag{5.13}$$

where Π_ω is the profit computed on workload ω, M_ω is the total number of processors, and L_ω is the number of hours of data provided in workload ω. We define *revenue per*

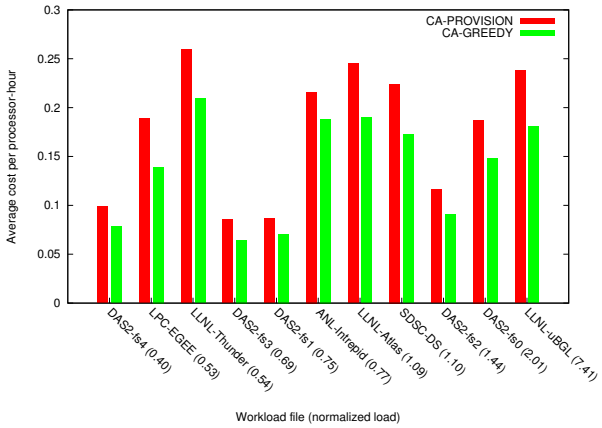


Figure 5.2: Average cost per processor-hour by CA-PROVISION and CA-GREEDY vs. normalized load.

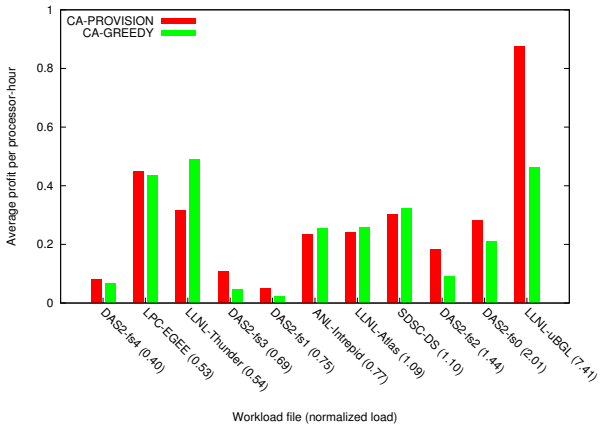


Figure 5.3: Average profit per processor-hour by CA-PROVISION and CA-GREEDY vs. normalized load.

processor-hour and *cost per processor-hour* in a similar fashion.

We plot the average revenue, the average cost, and the average profit per processor-hour versus the workload logs in Figures 5.1 to 5.3. In these figures, the workloads are sorted in ascending order of their normalized load. Note that the CA-PROVISION mechanism yields higher revenue in most of the cases. For workloads with normalized loads greater than 1.44, the revenue obtained by CA-PROVISION steadily increases exceeding that obtained by CA-GREEDY by up to 40%. This leads us to conclude that CA-PROVISION is capable of generating higher revenue where there is high demand for resources.

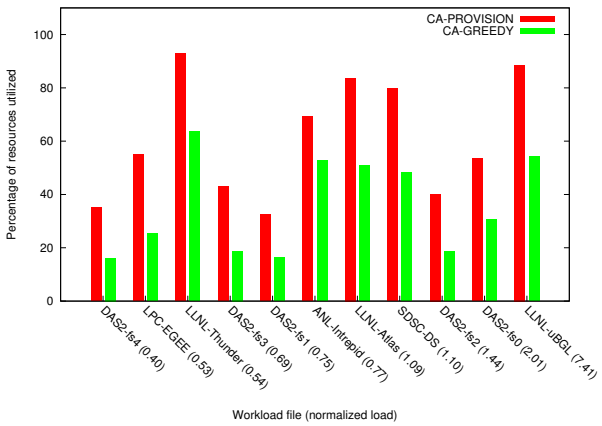


Figure 5.4: Resource utilization by CA-PROVISION and CA-GREEDY vs. normalized load

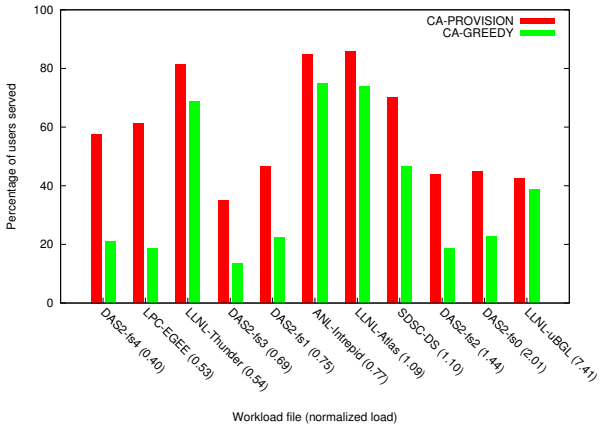


Figure 5.5: Percent users served by CA-PROVISION and CA-GREEDY vs. normalized load

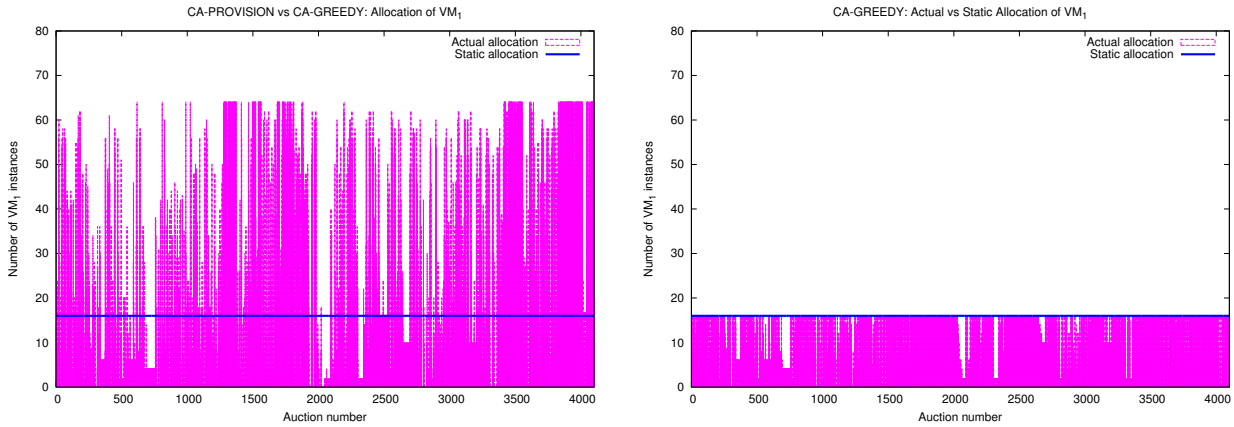
In Figure 5.2 we observe that CA-PROVISION incurs a higher total cost for all workloads. Since CA-PROVISION decides about the number of VMs dynamically, it can allocate a higher number of VM instances than CA-GREEDY in an auction with identical bidders. This explains the higher cost incurred by CA-PROVISION; a unit VM instance costs c_I per unit time when idle and $c_R > c_I$ per unit time while running (i.e., allocated to a user), as we assumed in Section 5.2. Therefore, by provisioning and allocating more VM instances, CA-PROVISION incurs higher cost to the cloud provider.

Now, the question is whether the interplay between increased revenue and increased

cost can generate a higher profit. Utilizing more resources means serving more customers hence selecting more bidders as winners. This interplay has two mutually opposite effects on the revenue. Obviously, increasing the number of winners has a positive effect on the revenue. On the other hand, selecting more winners pushes down their critical values, and thus, individual payments decrease. If the net effect is positive, we get a higher revenue and when it surpasses the increase in cost, we obtain a higher profit, and thus, achieve economies of scale. From Figure 5.3 we see that for normalized loads greater than 1.44, CA-PROVISION consistently generates higher profit than CA-GREEDY and the difference in profit grows rapidly. We also observe that for the workloads having load factors below 1.44 CA-PROVISION and CA-GREEDY obtain higher profit in equal number of cases. This suggests that for low loads the relative outcome of the mechanisms depends on other parameters.

In Figures 5.4 and 5.5 we compare the resource utilization and the percentage of served users obtained by the two mechanisms. CA-PROVISION achieves higher values for both utilization and percentage of served users. We want to draw the attention of the reader to the fact that in most of the cases the difference in utilization is around 30%. This is where we can improve a lot if we switch from static to dynamic provisioning and allocation. Since combinatorial auctions are already established tools for efficient allocation, combining them with dynamic provisioning can lead to a highly efficient resource allocation mechanism for clouds.

The number of users served is higher for CA-PROVISION because the VM instances are not statically provisioned. Therefore, a user requesting two VM_1 instances will not be left unallocated if there are no VM_1 instances available but a VM_2 instance is available as in the case of CA-GREEDY. Rather, CA-PROVISION 'sees' the available resource as a computing resource equivalent to two VM_1 instances and will allocate this, for instance, to a user bidding for two VM_1 instances or a user bidding for one VM_2 instance, depending on whose reported valuation is higher. This approach increases the number of users served by CA-PROVISION mechanism.

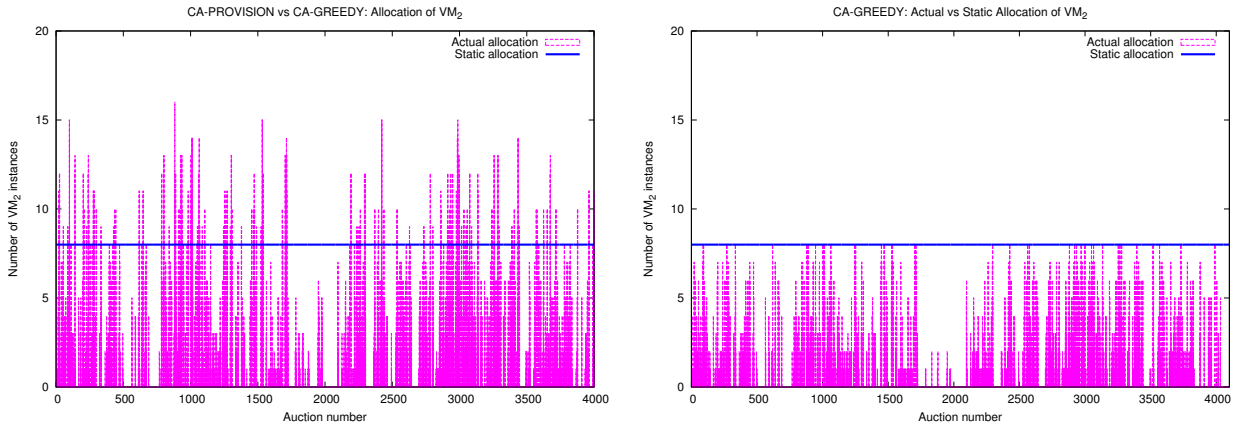


(a) (b)

Figure 5.6: Allocation of VM_1 instances: (a) by CA-PROVISION; (b) by CA-GREEDY. Workload file: DAS2-fs3-2003.

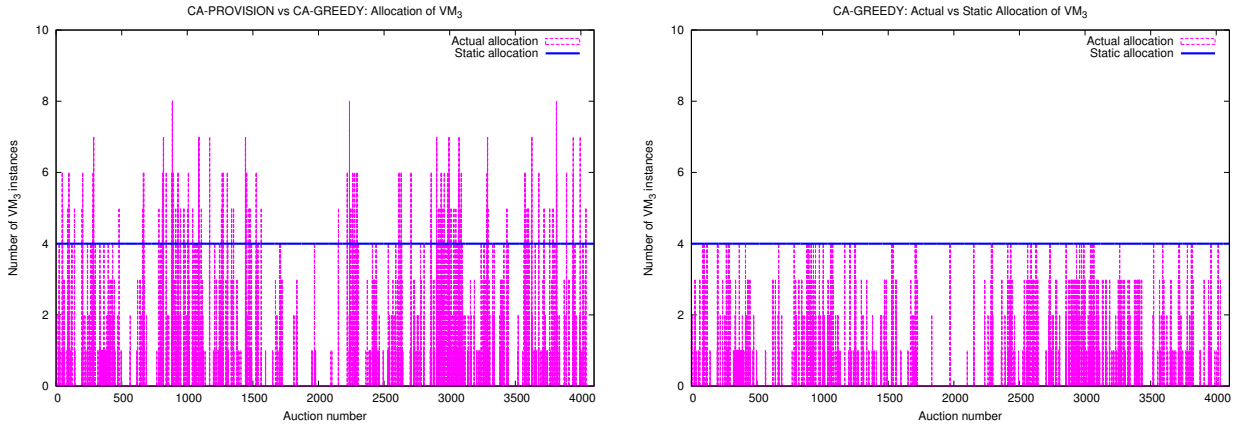
We now go into the details of the VM allocation by CA-GREEDY and CA-PROVISION for the DAS2-fs3-2003 workload. We pick a sample scenario from various combination of input parameters. In this experiment, the static VM allocation consists of 16 instances of type VM_1, 8 instances of type VM_2, 4 instances of type VM_3, and 2 instances of type VM_4, This is equivalent to 64 instances of unit size (i.e., type VM_1). For this workload, a total of 4100 auctions were held and in Figures 5.6 to 5.9, we show the allocation of different VM instances in all these auctions. The figures corresponding to the CA-PROVISION mechanism show the number of the VM instances that are provisioned by the mechanism as box plots. For example, in Figure 5.6a, we see that in many auctions, all 64 processors are configured as VM_1 instances. On the other hand, there are auction outcomes where no VM_1 instances are provisioned, as evident by the white strips touching the horizontal axis. The box plots in the figures corresponding to the CA-GREEDY mechanism show the number of the VM instances that are allocated to the users. In both categories of plots, we show the static allocation line to compare the differences between static and dynamic provisioning.

Figure 5.6a is particularly interesting because it shows that at times the demand for VM_1 goes far beyond what we would even think of allocating in advance. In some auctions



(a) (b)

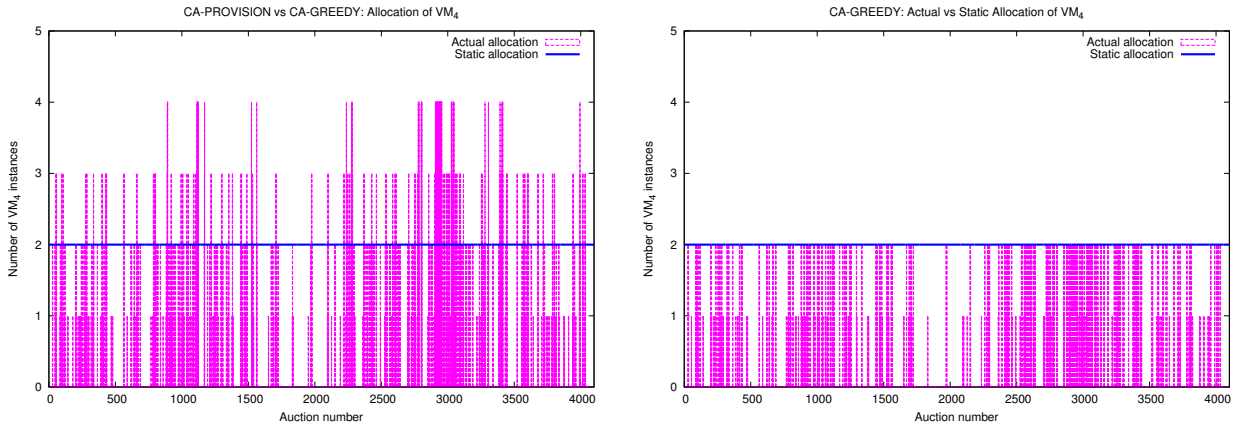
Figure 5.7: Allocation of VM_2 instances: (a) by CA-PROVISION; (b) by CA-GREEDY. Workload file: DAS2-fs3-2003.



(a) (b)

Figure 5.8: Allocation of VM_3 instances: (a) by CA-PROVISION; (b) by CA-GREEDY. Workload file: DAS2-fs3-2003.

demands for VM_1 instances are much higher and therefore they push the allocation to the boundary. On the other hand, if we compare it with Figure 5.6b, we see that CA-GREEDY indeed can capture the demand and allocate all sixteen available instances of VM_1 in most of the auctions, but is limited to the availability of statically provisioned VMs. Eventually it has to serve other less valued bids and looses revenue. Also, CA-GREEDY suffers from under-allocation as it is clear from Figures 5.8a and 5.8b. We see that the actual demand of



(a) (b)

Figure 5.9: Allocation of VM_4 instances: (a) by CA-PROVISION; (b) by CA-GREEDY. Workload file: DAS2-fs3-2003.

VM_3 instances is lower than what we allocate statically (Figure 5.8a) and the VM instances indeed remain unallocated in many cases (Figures 5.8b).

We can summarize the experimental results as follows. The CA-GREEDY mechanism is capable of generating higher revenue than CA-PROVISION when there is matching demand with the supply. Also, in an auction where items are not 'configurable' as in the case of cloud auctions, CA-GREEDY is a very efficient auction. But when we have reconfigurable items as in clouds, it is very hard to predict the demand very well in advance. In that case, CA-PROVISION is a better option and as today's technology supports, it can be deployed as a stand-alone configuration and allocation tool without much human intervention. There is also another use of this mechanism. One can combine CA-GREEDY and CA-PROVISION in a way that periodically CA-PROVISION will be executed to capture the current market demand, determine the static allocation that best matches the demand, and instantiate CA-GREEDY. If the utilization falls below a certain threshold, CA-PROVISION can be called to determine a good configuration again. This can also eliminate the need of detailed statistical analysis of demand to find an efficient static configuration for CA-GREEDY.

5.5 Summary

We addressed the problem of dynamically provisioning VM instances in clouds in order to generate higher profit, while determining the VM allocation with a combinatorial auction-based mechanism. We designed a mechanism called CA-PROVISION to solve this problem. We performed extensive simulation experiments with real workloads to evaluate our mechanism. The results showed that CA-PROVISION can effectively capture the market demand, provision the computing resources to match the demand, and generate higher revenue than CA-GREEDY, especially in high demand cases. In some of the low demand cases, CA-GREEDY performs better than CA-PROVISION in terms of profit but not in terms of utilization and percentage of served users. We conclude that an efficient VM instance provisioning and allocation system can be designed combining these two combinatorial auction-based mechanisms.

CHAPTER 6: AN ONLINE MECHANISM FOR DYNAMIC VM PROVISIONING AND ALLOCATION IN CLOUDS

6.1 Introduction

Virtualization technologies have created convenient ways for the cloud providers to allocate their computing resources. They define different configurations of virtual machines (VMs) and 'sell' the resources in units of VM instances. Currently, the cloud providers use fixed price-based mechanisms to allocate and sell VM instances (e.g., Windows Azure [40], Amazon EC2 [2]) or auction-based mechanisms to sell resources that are not utilized after the fixed-price based selling [3].

There are certain features of the problem of VM allocation in clouds that makes online mechanisms suitable for solving it. First of all, it is quite natural for users to submit bids for bundles of VMs rather than for individual VMs. For example, if a user requires one 'small' and one 'large' VM instance for a particular application, she would prefer to get both of them together, otherwise she would prefer nothing to getting only one of the VMs. Therefore, the allocation mechanism must allow users to express their preferences for a bundle of VMs and not only for individual VMs. On the other hand, the users submit their requests (or arrive at the system) continuously and would prefer to have their allocation decision as soon as possible, at the best possible price. Once they obtain the requested VM bundles, they would like to complete their entire task on the acquired resources.

Another consideration is that the cloud provider would prefer a mechanism that sup-

ports dynamic provisioning so that they can decide on the number of instances of different types of VMs based on the market demand. The cloud provider is also interested in maximizing its revenue or profit. Considering all the desired properties mentioned above, in this chapter, we design an online mechanism that provisions computing resources into VM instances, allocates them to users, and determines the payment for each user. Online mechanisms solve allocation problems without having all the information available. An online mechanism calculates allocation and payment as the participants arrive at the system and place their requests. This is particularly useful to guarantee allocation efficiency in systems where users arrive continuously and items being allocated are expiring items [47]. For the problem of VM instance allocation in clouds, the resources can be considered as expiring since the allocation is dependent on time: a resource not allocated at time t loses its utilization for that particular moment.

We design an online mechanism for provisioning and allocating VM instances in clouds. We design the bidding protocol so that a user requests a bundle of VMs expressing that she is only interested in the whole bundle and not a subset of it. Upon receiving such bids, the mechanism calculates the allocation and payment online. Also, the allocation cannot be preempted: a user receiving an allocation at time t continues to hold the resources for the time period she requested the resources for, while submitting her bid. We also provide theoretical results proving the properties of the proposed mechanism.

6.1.1 Our Contribution

We formulate the online VM provisioning and allocation problem and design an online mechanism to solve it. To the best of our knowledge, this is the first work that proposes an online mechanism for VM provisioning and allocation in clouds. We provide theoretical results that prove that the proposed mechanism is incentive compatible and runs in a reasonable amount of time. We perform extensive experiments and show that this mechanism improves the efficiency of allocation of VM instances in clouds.

6.1.2 Organization

The rest of the chapter is organized as follows. In Section 6.2, we formulate the problem of online VM provisioning and allocation. In Section 6.3, we introduce the basic concepts of mechanism design in general and online mechanism design in particular in the context of the problem we are addressing. In Section 6.4, we present our proposed online mechanism for VM provisioning and allocation. In Section 6.5, we characterize the properties of the proposed mechanism. In Section 6.6, we investigate the performance of the proposed mechanism by extensive simulation experiments. In summarize the research in Section 6.7.

6.2 VM Instance Allocation Problem

We consider a cloud provider that provisions its computing resources into m different types of VM instances. We denote these types of VM instances by $VM_1, \ldots, VM_m$. With each type VM_i we associate a 'weight' $w_i \in \mathbb{R}_+^*$ which specifies the relative computing power of VM_i with respect to the most powerful VM instance offered by the cloud provider (i.e., the instance of type VM_1). We consider, without loss of generality, that $w_1 = 1$ and $w_1 \leq \ldots \leq w_m$. The total power of the computing resources available for allocation is denoted by M, and is defined as the equivalent power of M instances of type VM_1. For example, let us consider a cloud provider that offers VM instances of three types: small (VM_1), medium (VM_2), and large (VM_3). The configurations of these VMs are as follows: $VM_1 \equiv$ (one 2GHz processor, 4GB memory, 1TB hard drive), $VM_2 \equiv$ (two 2GHz processors, 8GB memory, 2TB hard drive), and $VM_3 \equiv$ (four 2GHz processors, 16GB memory, 4TB hard drive). For this setting, the weights of the VMs would be $w_1 = 1$, $w_2 = 2$, and $w_3 = 4$. If, for example, the cloud provider has enough resources to provision 100,000 VM_1 instances, then $M = 100,000$. The goal of the cloud provider is to dynamically provision its available resources into VM instances and allocate them to users efficiently and at the same time maximize its profit.

There are n users $j = 1, \ldots, n$ who submit their requests (or 'bids') to the cloud

provider in order to secure bundles of VM instances for certain amount of time in order to execute their applications (jobs) on the cloud. A user j is characterized by her 'type' $\theta_j = (r_1^j, \ldots, r_m^j, a_j, l_j, d_j, v_j) \in \Theta_j$, where Θ_j is her type space. Here, $(r_1^j, \ldots, r_m^j)$ is the bundle of VM instances requested by user j, where $r_i^j \in \mathbb{N}$ is the number of VM_i instances requested by user j. We also denote the *bundle* by S_j and its *total weight* by $s_j = \sum_{i=1}^{m} w_i r_i^j$. The other components of user j's type θ_j are: $a_j \in \mathbb{N}$, the arrival time (i.e., the time at which user j submits her bid); $l_j \in \mathbb{N}$, the amount of time the requested bundle must be allocated for; and, $d_j \in \mathbb{N}$, the deadline for user j's job completion. We denote by δ_j, the time by which the bundle must be allocated to the user in order for her job to complete its execution. That is, $\delta_j = d_j - l_j$. The last component of θ_j is $v_j \geq 0$, the value user j receives if her requested bundle of VMs is allocated within time δ_j. We assume that the users are *single-minded*, i.e., each user j requires that either her requested bundle S_j be allocated to her and she pays for it, or she does not get any allocation and pays nothing.

In the following we give an example to clarify the meaning of the components of the user type. Suppose user j requires two VM_1 and one VM_3 instances for five hours to complete a particular application. Hence, she requests the bundle $S_j = (2, 0, 1)$ with a total weight $s_j = 2+4 = 6$. She submits her bid at time $t = 0$ and must get her job done by time $t = 10$. Hence, $a_j = 0$, $l_j = 5$, and $d_j = 10$. There is no use of the requested bundle if she gets it after $t = 5$, therefore $\delta_j = 5$. Finally, suppose the successful completion of the application yields a value of \$50 to her, hence the valuation of her bundle is $v_j = 50$. Altogether, user j's type is $\theta_j = (2, 0, 1, 0, 5, 10, 50)$. We also call it her 'true bid', because she may choose to report a different type from her type space (i.e., all possible combinations of the values) to the cloud provider if she benefits by doing so.

The cloud provider allocates bundles of VM instances to users by dynamically provisioning the available resources and charges them money. The provider must decide the allocation of a bundle S_j within the time interval $[a_j, \delta_j]$. If a user is not granted the allocation within this interval, her request is declined and she withdraws her request. Formally, the cloud provider computes an allocation set $A \subseteq \Theta \times \mathbb{N}$ and a payment set $P \subseteq \Theta \times \mathbb{R}_+$.

A tuple $(\theta_j, t_j) \in A$ represents that bundle S_j of user j of type θ_j has been allocated at time t_j, where $a_j \leq t_j \leq \delta_j$. A tuple $(\theta_j, p_j) \in P$ says that user j of type θ_j has to pay p_j to the cloud provider. For the above user j, if she is granted the allocation at time $t = 3$ and charged \$30, we would have $(\theta_j, 3) \in A$ and $(\theta_j, 30) \in P$. Users who do not get the requested allocation pay zero.

We also consider that the cloud provider decides about the provisioning and allocation online, i.e., whenever some users and resources are available. Once a bundle S_j is allocated to a user j, it will not be reclaimed until l_j units of time after the allocation. Due to the limited resources, the cloud provider cannot allocate the VM bundles to all users at any given time. Also, unallocated users are going to potentially leave the system when their respective δ_j time has passed and they will not contribute to the provider's revenue. Therefore, the challenge of the cloud provider is to make provisioning and allocation decisions dynamically while trying to maximize its profit.

Since very little is known about revenue maximization in the mechanism design literature, mechanisms are usually designed with the goal of maximizing the sum of the valuations of the users [44]. Thus a reasonable goal is to allocate the VM instances so that the sum of the valuations of the users who receive their bundles is maximized. We formulate the *Online VM Provisioning and Allocation Problem (OVMPA)* as follows.

$$\max \sum_{j:(\theta_j,t_j)\in A} v_j \tag{6.1}$$

subject to:

$$\sum_{j:\theta_j \in \tilde{N}^{(t)}} s_j \leq M \tag{6.2}$$

Equation (6.1) is the objective of the problem, that is, the cloud provider maximizes the sum of the values of the users who obtain their requested bundles. The constraint in equation (6.2) says that at any given time t the allocation is limited to M, the total

amount of resources available. Here,

$$\tilde{N}^{(t)} = \{j \mid (\exists t_j \leq t : (\theta_j, t_j) \in A) \wedge (t_j + l_j > t)\}$$

is the set of users who have been allocated prior to or at t and have yet to complete their allotted time slot at time t.

A straight-forward solution to the above problem would be to select users with the highest values, and then charge them their reported values. However, the users are rational and they may misreport their types if it benefits them to do so. For example, a user may report a lower valuation to pay less or a higher valuation to enhance her chance of winning. If the cloud provider prefers jobs with an earlier deadline while breaking a tie for values, users may choose to report a deadline that is earlier than their actual deadline. Since this information is private, the provider needs to employ a mechanism to compute the allocation and payment based on the users' reported values in such a way that the system-wide goals set by the provider are achieved.

Therefore, the problem of designing an online mechanism for solving the OVMPA is as follows. Design an online mechanism that computes an allocation set A and payment set P on the problem space $\Theta = \Theta_1 \times \ldots \times \Theta_n$ to maximize the objective function in Equation (6.1) satisfying the constraint given in Equation (6.2). The information reported by the users to the mechanism is denoted as $\hat{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_n) \in \Theta$, where $\hat{\theta}_j \in \Theta_j$. The goal of the mechanism is to compute an efficient allocation even if $\hat{\theta}_j \neq \theta_j$ and calculate payments in a way so that it provides incentives to the users to report their true types.

6.3 Online Mechanism Design Framework

A mechanism $\mathcal{M} = \{\mathcal{A}, \mathcal{P}\}$ is a set of functions $\mathcal{A}$ for computing the allocation and $\mathcal{P}$ for computing the payment for each user. Here, $\mathcal{A} : \Theta \rightarrow A$ and $\mathcal{P} = (p_1(.), \ldots, p_n(.))$, where $p_j : \Theta \rightarrow \mathbb{R}$ for $j = 1, \ldots, n$. In the context of OVMPA, the allocation function $\mathcal{A}$ computes

the allocation set $A \subseteq \Theta \times \mathbb{N}$ from the bids reported by the users. The allocation set A is the set of the tuples (θ_j, t_j), where j is the user receiving her requested bundle at time t_j. Function $p_j(.)$ determines user j's payment based on the bids of all users.

Each user j is characterized by a *valuation function* V_j defined as follows:

$$V_j(\mathcal{A}(\hat{\theta}), \theta_j) = \begin{cases} v_j & \text{if}(\theta_j, t_j) \in A \wedge t_j \leq d'_j \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

This means that the user receives the value v_j if she secures the requested bundle, and no value, otherwise. We quantify user j's benefit through a *utility function* defined as the difference between the value she receives from the mechanism and the payment charged to her:

$$U_j(\mathcal{A}(\hat{\theta}), \theta_j) = V_j(\mathcal{A}(\hat{\theta}), \theta_j) - p_j(\hat{\theta}) \tag{6.4}$$

In the example presented above user j derives a utility of \$20 if she values the bundle at \$50, receives her bundle at time $t \leq 5$, and pays \$30 for it (as shown in Section 6.2). If the mechanism decides not to allocate her the requested bundle, she receives a value of zero. In that case, if the mechanism does not charge her any payment, her utility will be zero as well.

We are interested in designing mechanisms which have two important properties, *incentive compatibility* and *individual rationality*. In the following, we denote by $\hat{\theta}_{-j} = (\hat{\theta}_1, \ldots, \hat{\theta}_{j-1}, \hat{\theta}_{j+1}, \ldots, \hat{\theta}_n) \in \Theta_{-j}$ the bids of all users except j. Recall that the true type of user j is θ_j and her reported type (or bid) $\hat{\theta}_j$ may not be equal to θ_j.

Definition 4 (Incentive compatibility)**.** *A mechanism $\mathcal{M}$ is* incentive compatible *if for all j, $\hat{\theta}_j \in \Theta_j$, and $\hat{\theta}_{-j} \in \Theta_{-j}$,*

$$U_j(\mathcal{A}(\theta_j, \hat{\theta}_{-j}), \theta_j) \geq U_j(\mathcal{A}(\hat{\theta}_j, \hat{\theta}_{-j}), \theta_j)$$

That is, the users maximize their utilities by reporting their true types to the mecha-

nism, irrespective of the other users' bids. This is a very important property, because if satisfied, the users participating in the mechanism will not have incentives to report other types than their true types to the mechanism. That is, truthful reporting is their best strategy.

Definition 5 (Individual rationality). *A mechanism is* individually rational *if a user never incurs a loss by reporting her true type. Formally, for all j, true type $\theta_j \in \Theta_j$, and $\hat{\theta}_{-j} \in \Theta_{-j}$,*

$$U_j(\mathcal{A}(\theta_j, \hat{\theta}_{-j}), \theta_j) \geq 0$$

That is, regardless of other users' bids, a user reporting her type truthfully will never obtain a negative utility by participating in the mechanism. This is a very important property of a mechanism since it encourages users to voluntarily participate in the mechanism.

In order to obtain an incentive compatible mechanism the allocation function $\mathcal{A}$ must be *monotone* and the payment function $\mathcal{P}$ must implement the *critical value payment* [43].

We introduce a preference relation, $\succeq$, on the set of user bids. For example, $\hat{\theta}'_j \succeq \hat{\theta}_j$ means that bid $\hat{\theta}'_j$ is more preferred to the mechanism than bid $\hat{\theta}_j$. In our context $\hat{\theta}'_j \succeq \hat{\theta}_j$ if $\hat{s}'_j \leq \hat{s}_j$, $\hat{a}'_j \leq \hat{a}_j$, $\hat{l}'_j \leq \hat{l}_j$, $\hat{d}_j \geq \hat{d}_j$ and $\hat{v}'_j \geq \hat{v}_j$.

Definition 6 (Monotone allocation function). *An allocation function $\mathcal{A}$ is monotone if for all j, $\hat{\theta}_j, \hat{\theta}'_j \in \Theta_j$, and $\hat{\theta}_{-j} \in \Theta_{-j}$,*

$$(\hat{\theta}'_j \succeq \hat{\theta}_j) \wedge (\exists t_j : (\hat{\theta}_j, t_j) \in \mathcal{A}(\hat{\theta}_j, \theta_{-j})) \Rightarrow \exists t'_j : (\hat{\theta}'_j, t'_j) \in \mathcal{A}(\hat{\theta}'_j, \theta_{-j})$$

This means that if user j gets her requested bundle by declaring type $\hat{\theta}_j$, she will also get the resources by declaring type $\hat{\theta}'_j$, where type $\hat{\theta}'_j$ is more preferred over $\hat{\theta}_j$.

Definition 7 (Critical value payment). *Critical value v_j^c for user j is defined as*

$$v_j^c = \arg\min_{v'_j \geq 0} (\exists t_j : (\hat{\theta}_j = (r_1^j, \ldots, r_m^j, a_j, l_j, d_j, v'_j), t_j) \in \mathcal{A}(\hat{\theta}_j, \theta_{-j}))$$

This means that the critical value for a user is the minimum amount she needs to report to the mechanism in order to receive her requested bundle. The payment function should charge a user her critical value in order to obtain an incentive compatible mechanism.

The challenges of designing online mechanisms come from the fact that the mechanisms do not have full information on the requests and from the richer dimension of user types. In the context of OVMPA, the users can misreport their arrival time (i.e., submit bid at a different time than the actual time when the task is available) to gain higher chances of allocation or lower payments. For example, the critical value of a user may be lower if she submits her bid at a later time. Therefore, the mechanism must ensure that users do not gain by misreporting the arrival time as well as other parameters. Misreporting the other parameters may affect the allocation and payment for both online and offline mechanisms. In the following section, we design an online mechanism that solves OVMPA while addressing the above challenges.

6.4 Online Mechanism for VM Allocation

We now present our mechanism for Online VM Provisioning and Allocation (MOVMPA) that solves the OVMPA problem.

6.4.1 Mechanism MOVMPA

The mechanism MOVMPA is structured as an event handler, which is invoked when a new bid arrives or a user completes her time for the allocated bundle and releases the VM instances to the provider. We assume that the information about the users and the resources is made available to the mechanism via some standard protocol. MOVMPA uses this information to determine the set of users and resources available for allocation at the current time and calls the allocation and the payment functions. We present the MOVMPA mechanism in Algorithm 7.

Algorithm 7 Mechanism MOVMPA

Require: $Event, A, P$
Ensure: A, P
1: $t \leftarrow$ Current time
2: $N^{(t)} \leftarrow \{\hat{\theta}_j \in N \mid \hat{a}_j \leq t \wedge \neg\exists t_j < t : (\hat{\theta}_j, t_j) \in A\}$
3: $\tilde{N}^{(t)} \leftarrow \{\hat{\theta}_j \mid \exists t_j < t : (\hat{\theta}_j, t_j) \in A \wedge t_j + \hat{l}_j > t\}$
4: $M^{(t)} \leftarrow M - \sum_{j:\hat{\theta}_j \in \tilde{N}^{(t)}} \hat{s}_j$
5: **if** $M^{(t)} = 0 \vee N^{(t)} = \emptyset$ **then return**
6: $R \leftarrow M^{(t)}$
7: $A^{(t)} \leftarrow$ MOVMPA-ALLOCATE$(t, N^{(t)}, R)$
8: $A \leftarrow A \cup A^{(t)}$
9: $P^{(t)} \leftarrow \{(\hat{\theta}_j, \hat{v}_j) \mid (\hat{\theta}_j, t_j) \in A^{(t)}\}$
10: $P \leftarrow P \cup P^{(t)}$
11: $P \leftarrow$ MOVMPA-PAYMENT$(t, P, N^{(t)}, \tilde{N}^{(t)}, M^{(t)})$
12: $N^P \leftarrow \{\hat{\theta}_j \in N \mid t' < \delta_j \leq t\}$
13: **for** each $\hat{\theta}_j \in N^P$ **do**
14: **if** $\exists p_j : (\hat{\theta}_j, p_j) \in P$ **then**
15: user j pays p_j
16: **else**
17: user j pays 0
18: **end if**
19: **end for**
20: $t' \leftarrow t$
21: **return** A, P

MOVMPA takes as input the event, the current allocation set and the payment set. An event is either a release of resource or an arrival of a user request. We assume that the system stores these two sets and passes them to MOVMPA when it is invoked. MOVMPA updates the sets and returns back to the system. In lines 1-4, MOVMPA sets the current time to t and initializes three variables as follows: $N^{(t)}$, the set of bids of the users that have not been allocated so far; $\tilde{N}^{(t)}$, the set of bids of the users that have been allocated in the past and have not finished their time of allocation; and, $M^{(t)}$, the total weight of the resources that are available for allocation at time t. The mechanism proceeds only if resources and users are available. It calls the allocation function MOVMPA-ALLOCATE with the user bids that have not been allocated yet and the resources that are available at time t. MOVMPA-ALLOCATE returns set $A^{(t)}$, the set of users who would receive their

requested bundles at time t (line 7).

Next, MOVMPA updates the overall allocation set A to $A^{(t)}$. The bids in $A^{(t)}$ are also inserted into the payment set, with $p_j = \hat{v}_j$ as their initial payment. However, this payment is updated by calling the payment function MOVMPA-PAYMENT (line 11). In fact, the payment of user j is going to be updated several times until $t = \delta_j$, i.e., until the time instance the user must receive allocation of the bundles she requires. Thus, MOVMPA-PAYMENT calculates the payments for these users and updates the payment set P. The next step for MOVMPA is to determine the set N^P of the bids of users j for whom the current time has gone past their respective δ_j times (line 12). However, this set only includes the users whose δ_j has passed after t', where t' is the time of the last invocation of MOVMPA (line 12). If user j has been already provided with her bundle, her payment will no longer change and the payment that is up to date at the current time will be her final payment charged to her. If user j has not received her requested bundle until t and $t > \delta_j$, she will not be successful in getting an allocation to get her job done. In this case, user j will pay $p_j = 0$ (lines 14-19). The computation of the prices will be presented when we discuss the MOVMPA-PAYMENT function.

6.4.2 Allocation function

The allocation function MOVMPA-ALLOCATE is given in Algorithm 8. In order to describe this function we define a new parameter called 'bid density', $\rho_j = \frac{\hat{v}_j}{\hat{s}_j \times \hat{l}_j}$. The intuition behind bid density is as follows. We can reformulate the VM allocation problem as the problem of allocating rectangles in a two-dimensional space of VM weight and time. The bid by user j for a bundle of VM instances of weight $\hat{s}_j$ for time $\hat{l}_j$ can be interpreted as requesting a rectangle with area $\hat{s}_j \times \hat{l}_j$ in that two-dimensional space, and user j values this area at $\hat{v}_j$. Hence, ρ_j is how much user j values a 'unit area' of the rectangular space. Obviously, the cloud provider is interested in users who want to pay more per unit of their resources per unit time. MOVMPA-ALLOCATE uses ρ_j to determine the relative values of the bids.

Algorithm 8 MOVMPA-ALLOCATE

Require: $t, N^{(t)}, R$
Ensure: $A^{(t)}$
1: $A^{(t)} \leftarrow \emptyset$
2: Sort all $\hat{\theta}_j \in N^{(t)}$ in non-increasing order of ρ_j
3: **for** each $\hat{\theta}_j \in N^{(t)}$ in sorted order **do**
4: **if** $s_j \leq R$ **then**
5: $A^{(t)} \leftarrow A^{(t)} \cup (\hat{\theta}_j, t)$
6: $R \leftarrow R - \hat{s}_j$
7: **end if**
8: **end for**
9: **return** $A^{(t)}$

First, MOVMPA-ALLOCATE sorts all bids in non-increasing order of ρ_js. Ties are broken in the following order: prefer earlier δ_j, smaller $\hat{l}_j$, and then smaller $\hat{s}_j$. Further ties are broken arbitrarily. Then the algorithm allocates bundles requested by these users while resources last. Finally, it returns the set $A^{(t)}$ of users who are selected for allocation at time instance t.

MOVMPA-ALLOCATE simply tries to maximize the sum of the reported valuations of the users who would be granted their requested bundles. In the case of a tie, by giving priority to users with a smaller δ_j, (i.e., users who need to leave the system earlier if they don't get their bundles), it also makes sure that the highest possible number of users are served. For the same reason, in case of a tie with δ_j, priority is given to users who request the resources for a smaller amount of time.

6.4.3 Payment function

We give the payment function MOVMPA-PAYMENT in Algorithm 9. The payment function requires as input the current time t, the payment set P, the amount of resources available before calling the allocation function $M^{(t)}$, the set of users who were considered at the allocation function $N^{(t)}$, and, the set of users who are occupying resources at the current time t, $\tilde{N}^{(t)}$. It is worth mentioning that users that were granted the requested bundle at time t also belong to the set $N^{(t)}$.

Algorithm 9 MOVMPA-PAYMENT

Require: $t, P, N^{(t)}, \tilde{N}^{(t)}, M^{(t)}$
Ensure: P
sort all $\hat{\theta}_j \in N^{(t)}$ in non-increasing order of ρ_j
for each $(\hat{\theta}_j, p_j) \in P : \delta_j \leq t$ **do**
  $N'^{(t)} \leftarrow N^{(t)} \setminus \hat{\theta}_j$
  if $\hat{\theta}_j \in \tilde{N}^{(t)}$ **then**
    $R' \leftarrow M^{(t)} + \hat{s}_j$
  else
    $R' \leftarrow M^{(t)}$
  end if
  for each $\hat{\theta}_{j'} \in N^{(t)}$ **do**
    $\bar{v}_j \leftarrow \rho_{j'} \times \hat{s}_j \times \hat{l}_j$
    $\bar{\theta}_j \leftarrow (r_1^j, \ldots, r_m^j, t, \hat{l}_j, \hat{d}_j, \bar{v}_j)$
    $A' \leftarrow$ MOVMPA-ALLOCATE$(t, N'^{(t)} \cup \{\bar{\theta}_j\}, R')$
    if $(\bar{\theta}_j, t) \in A'$ **then**
      $p_j \leftarrow \min(p_j, \bar{v}_j)$
    end if
  end for
  $\bar{\theta}_j \leftarrow (r_1^j, \ldots, r_m^j, t, \hat{l}_j, \hat{d}_j, 0)$
  $A' \leftarrow$ MOVMPA-ALLOCATE$(t, N'^{(t)} \cup \{\bar{\theta}_j\}, R')$
  if $(\bar{\theta}_j, t) \in A'$ **then**
    $p_j \leftarrow 0$
  end if
end for
return P

The main idea of MOVMPA-PAYMENT is to calculate the critical payment of each user j with $\delta_j \leq t$ as if their time of arrival is t. By repeatedly calling this function at each event, MOVMPA ensures that the critical payment of a user j is calculated every time an event occurs between $\hat{a}_j$ and δ_j. Formally, MOVMPA-PAYMENT calculates the critical value

$$v_j^{ct} = \arg\min_{v'_j \geq 0}(\exists t_j : (\hat{\theta}'_j = (\hat{r}_1^j, \ldots, \hat{r}_m^j, t, \hat{l}_j, \hat{d}_j, v'_j), t_j) \in \mathcal{A}(\hat{\theta}_j, \theta_{-j})) \tag{6.5}$$

at time t, for all users j with $\delta_j \geq t$. Based on this critical value the MOVMPA mechanism computes the critical value as

$$v_j^c = \min_{t \in [\hat{a}_j, d'_j]} v_j^{ct} \tag{6.6}$$

Table 6.1: User bids

$\hat{\theta}_j$	$\hat{s}_j$	$\hat{a}_j$	$\hat{l}_j$	$\hat{d}_j$	$\hat{v}_j$	δ_j	ρ_j
$\hat{\theta}_1$	3	0	3	5	5	2	0.56
$\hat{\theta}_2$	3	0	3	4	4	1	0.44
$\hat{\theta}_3$	2	1	5	8	6	3	0.60
$\hat{\theta}_4$	2	1	2	5	3	3	0.75
$\hat{\theta}_5$	3	3	4	9	8	5	0.67
$\hat{\theta}_6$	3	3	6	10	9	4	0.50

That is, v_j^c is the minimum value user j must report to get her requested bundle for any arrival time $a'_j \in [\hat{a}_j, d'_j]$.

MOVMPA-PAYMENT considers users from the payment set that have $\delta_j \leq t$. For each user j, the arrival time component of her type is set to t and her value is set to the values of each user to find the minimum value to be reported by user j in order to get her requested bundle (lines 2-16). If no such minimum value is found, the payment is set to zero (lines 17-21). One can also set this value to a predefined reserve price. Finally, the updated set P is returned to the mechanism.

The mechanism keeps updating P by calling MOVMPA-PAYMENT and charges the updated payment at time t to users j for which $\delta_j < t$. Users who were not allocated any resources pay zero.

Example 1. We show the execution of the mechanism by considering a setting in which the users bid as shown in Table 6.1. For example, user 1's bid $\hat{\theta}_1$ contains the following information: the weight of her requested bundle is $\hat{s}_1 = 3$, she submits her bid at $\hat{a}_1 = 0$, she requests the bundle for $\hat{l}_1 = 3$ time units, her deadline is $\hat{d}_1 = 5$, and she values the allocation of the bundle for the entire time at $\hat{v}_1 = 5$. We also show for each user, the value of $\delta_j = \hat{d}_j - \hat{l}_j$, the time by which the bundle must be allocated to meet the deadline, and $\rho_j = \frac{\hat{v}_j}{\hat{s}_j \times \hat{l}_j}$, the bid density.

We show the execution of MOVMPA for this setting in Table 6.2. The execution is shown as a time diagram, where the respective sets $N^{(t)}$, $\tilde{N}^{(t)}$, $M^{(t)}$, A, P, and N^P are shown for each time t. As a reminder, $N^{(t)}$ is the set of bids of users that participate at time

Table 6.2: Execution of MOVMPA

t	$t = 0\_$	$t = 0$	$t = 1$	$t = 2$	$t = 3$
$N^{(t)}$	$\emptyset$	$\{\theta_1, \theta_2\}$	$\{\theta_4, \theta_3, \theta_2\}$	$\{\theta_3\}$	$\{\theta_5, \theta_3, \theta_6\}$
$\tilde{N}^{(t)}$	$\emptyset$	$\{\theta_1\}$	$\{\theta_4, \theta_1\}$	$\{\theta_4, \theta_1\}$	$\{\theta_5, \theta_3\}$
$M^{(t)}$	5	2	0	0	0
A	$\emptyset$	$\{(\theta_1, 0)\}$	$\{(\theta_1, 0), (\theta_4, 1)\}$	$\{(\theta_1, 0), (\theta_4, 1)\}$	$\{(\theta_1, 0), (\theta_4, 1), (\theta_5, 3), (\theta_3, 3)\}$
P	$\emptyset$	$\{(\theta_1, 4)\}$	$\{(\theta_1, 4), (\theta_4, 2.4)\}$	$\{(\theta_1, 4), (\theta_4, 2.4)\}$	$\{(\theta_1, 4), (\theta_4, 2.4), (\theta_5, 6), (\theta_3, 0)\}$
$N^{(P)}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{\theta_2\}$	$\{\theta_4, \theta_1\}$

t, $\tilde{N}^{(t)}$ is the set of bids of users that are holding some resources at time t (including those who win their bids at time t), $M^{(t)}$ is the amount of resources available after allocation at time t, A and P are the allocation and payment sets, and N^P is the set of users whose payments are 'finalized' at time t. We assume that the sum of the VM weights in this example is $M = 5$.

In the second column, the initial value of all the variables are shown, where $M^{(t)} = M = 5$ and all set variables are empty. At $t = 1$, users 1 and 2 submit their bids and hence $N^{(t)} = \{\theta_1, \theta_2\}$. Since $\rho_1 > \rho_2$, user 1 is given the allocation of size $\hat{s}_1 = 3$. The remaining resources $M^{(t)} = 2$ are not sufficient to allocate user 2 ($\hat{s}_2 = 3$), so she loses at time $t = 0$. User 1's payment is computed as $p_2 = \rho_2 \times \hat{s}_1 \times \hat{l}_1$, since user 2 needs to bid at least this amount to defeat user 2 and get the allocation. The sets $\tilde{N}^{(t)}$, A, and P are updated to reflect the current allocation. At time $t = 1$, users 3 and 4 submit their bids, where $\rho_4 > \rho_3 > \rho_2$. The set $N^{(t)}$ shows the bids in order of non-increasing ρ_j values. Here we see that the new winner is 4 and user 1 is still holding the resources allocated to her, since she requires them for three time units. At time $t = 2$, θ_2 is discarded from the bidding users set $N^{(t)}$, because $t > \delta_2 = 1$ and she cannot meet her deadline. θ_2 is moved to N^P to finalize her payment $p_j = 0$. Note that here no auction is executed since the available resources are still zero.

At $t = 3$, both users 1 and 4 complete their allocated time. Their bids are moved to N^P and their current payments are set as their final payments. In this example, their payments did not change in any subsequent auction after they received the allocation. Users 5 and 6 arrive in the system and users 5 and 3 win their bundles at this time. User 3 pays zero

because she requests a bundle with $\hat{s}_j = 2$, which matches with the remaining resources. User 6 requests more than what is available at time $t = 3$, therefore user 3 would win even if she would bid zero. In a scenario with reserve price, user 3's payment will be set to the reserve price.

6.5 Properties of MOVMPA

In this section, we prove that the MOVMPA mechanism is incentive compatible and individually rational. We also perform competitive analysis and determine the runtime complexity of the mechanism.

Theorem 4. *MOVMPA is an individually rational mechanism.*

Proof. Let user j declare her true type θ_j to the mechanism. If she gets her requested bundle, then she pays $p_j \leq v_j$, because initially she is assigned the payment equal to v_j, and it is updated at each invocation of the mechanism until the time becomes d'_j. The update takes the minimum of the payment computed so far, therefore it will be always that $p_j \leq v_j$. Therefore, her utility $U_j = v_j - p_j \geq 0$. On the other hand, if she does not win her bundle, her valuation and payment are zero and hence the utility is zero. □

We prove the following lemmas and use them to prove that MOVMPA is incentive compatible.

Lemma 3. *User j can only misreport a $\hat{\theta}_j$ with $\hat{r}_i^j \geq r_i^j$ for all i, $\hat{a}_j \geq a_j$, $\hat{l}_j \geq l_j$, and $\hat{d}_j \leq d_j$.*

Proof. The true arrival time of a user into the system is when her intended task is ready to execute. There is no reason for a user to submit her request earlier than when the application is ready for execution. On the other hand, reporting a lower number of VM instances or a low required time will not allow the user to execute and complete her application. Similarly, reporting a later deadline may result in getting the bundle too late to complete the desired application in time. □

Lemma 4. *Let $\hat{\Theta}_j \subset \Theta_j$ be the type space of possible types user j may report to the mechanism, according to Lemma 3. Then for each $\hat{\theta}'_j, \hat{\theta}_j \in \Theta_j$, $\hat{\theta}'_j \succeq \hat{\theta}_j$, if $\exists t_j : (\hat{\theta}_j, t_j) \in \mathcal{A}(\hat{\theta}_j, \Theta_{-j})$, then $\exists t'_j : (\hat{\theta}'_j, t'_j) \in \mathcal{A}(\hat{\theta}'_j, \Theta_{-j})$. In other words, if user j wins by bidding $\hat{\theta}_j$, then she will also win if she reports a more preferable bid.*

Proof. Clearly, if user j reports $\hat{v}'_j \geq \hat{v}_j$, her bid $\hat{\theta}'_j$ will be allocated if $\hat{\theta}_j$ is also allocated. Similarly, if a user gets the allocation by reporting $\hat{d}_j$, she will also get it by reporting $\hat{d}'_j \geq \hat{d}_j$. Proofs for the other parameters follow the same logic, based on the priorities set for ordering the reported types in MOVMPA-ALLOCATE (Subsection 6.4.2). □

Lemma 5. *Suppose user j wins her bid by reporting both $\hat{\theta}_j$ and $\hat{\theta}'_j$, where they differ only on the arrival time $\hat{a}'_j \geq \hat{a}_j$, then their payment $p'_j \geq p_j$.*

Proof. Since the mechanism calculates the critical value payment for all $a'_j \in [\hat{a}_j, d'_j]$, the minimum calculated over the range $[\hat{a}_j, d'_j]$ must be less than or equal to the minimum calculated over the range $[\hat{a}'_j, d'_j]$. □

Lemma 6. *If user j gets the requested allocation by bidding both $\hat{\theta}_j$ and $\hat{\theta}'_j$, where the types differ only on the valuation, $\hat{v}_j \neq \hat{v}'_j$, then they pay the same amount.*

Proof. Since we compute the minimum value that the users must report to get the allocation, the minimum value is the same for both $\hat{\theta}_j$ and $\hat{\theta}'_j$, given that the other users' reported types remain the same. Hence, j pays the same amount for reporting both types. This is the critical value payment that we calculate in MOVMPA-PAYMENT. □

Theorem 5. *MOVMPA is an incentive compatible mechanism.*

Proof. Lemmas 3 and 4 show that the allocation algorithm is monotone. Lemmas 5 and 6 show that the mechanism implements the critical payment. Hence, MOVMPA is incentive compatible. □

Theorem 6. *The time complexity of MOVMPA is $O(n \log n)$.*

Proof. The costliest operation in MOVMPA-ALLOCATE is sorting the users in line 2. This requires a runtime of $O(n \log n)$. However, n is the total number of users and each time the mechanism encounters only a fraction of them.

MOVMPA-PAYMENT seems rather complicated because it repeatedly calls MOVMPA-ALLOCATE. But in an actual implementation, we do not need to call the allocation function to determine the critical payment. For each user j that is a winner, we need to consider only the users j' who lose at time t, in non-increasing order of their bids, and check whether user j' can win her bid if j does not participate. This check is done in a single comparison of resources occupied by j and the resources requested by j'. When we find first such j', user j's payment is $\hat{v}_{j'}$. We need to sort the losing bids only once at the beginning of the algorithm, which again has a running time of $O(n \log n)$, where n is a much larger number than what that algorithm encounters in practice. We present the payment algorithm in this fashion in order to show the underlying concepts. □

Now we prove the competitive ratio of MOVMPA. The *competitive ratio* of an online algorithm is $c > 1$ if the ratio of its performance to an optimum offline algorithm is $1/c$. We prove the competitive ratio of MOVMPA by choosing an input that produces the worst-case scenario for MOVMPA.

Theorem 7. *MOVMPA mechanism has a competitive ratio of M.*

Proof. Consider two bids θ_1 and θ_2 where $s_1 = 1$, $s_2 = M$, $(a_1, l_1, d_1, v_1) = (0, l, l, v)$, and $(a_2, l_2, d_2, v_2) = (1, l, l + 1, Mv + \epsilon)$, where $l > 1$. MOVMPA will allocate user 1 her requested bundle that she will release at time $l > 1$. Now, since for θ_2, $a_2 + l_2 = 1 + l = d_2$, user 2 must get the allocation at time $a_2 = 1$, otherwise she withdraws her bid because she cannot satisfy her deadline. Therefore, MOVMPA will allocate VM instances only to user 1 and obtain a value of v (Equation (6.1)). On the other hand, an optimum offline algorithm will allocate resources to user 2 and obtain a value of $Mv + \epsilon > M \times v$. Hence, the competitive ratio of MOVMPA is M. □

6.6 Experimental Results

In this section, we evaluate MOVMPA through simulation experiments. We compare MOVMPA with a good offline mechanism to identify its strengths and weaknesses. We generate user bids from real workload data, which are then parallelly submitted to both mechanisms. Of the experiment outcome, we mainly focus on three parameters: percent of users served (i.e., those who received their requested bundle for the entire bundle required), average revenue generated per served user, and average utility received by each served user. These three metrics are most important to evaluate a mechanism, because they determine the resource utilization, provider's profit, and user satisfaction.

6.6.1 Experimental Setup

We choose the offline mechanism CA-PROVISION from the previous chapter. to compare with MOVMPA. As we recall, CA-PROVISION solves the dynamic VM provisioning and allocation problem in clouds. It is invoked at regular intervals, (e.g., hourly). When it is invoked, it considers all the bids collected during the past interval and runs a combinatorial auction to determine the set of winners, their payments, and the number of different VM instances to provision. The VM instances are allocated for one time interval and users requiring subsequent access must continue to bid until their time requirement is fulfilled. A bundle allocated to a user at current period may be preempted later if the demand increases.

We choose CA-PROVISION to compare with MOVMPA because of two reasons. First, it is the closest mechanism to MOVMPA in the literature and the major difference between the two mechanisms are that one of them are online and the other is offline. Another reason for choosing CA-PROVISION is that we compared it with another successful mechanism CA-GREEDY and found that CA-PROVISION performs better in different aspects. This leads us to the decision that comparing with CA-PROVISION will correctly position MOVMPA in VM allocation mechanisms in terms of different performance metrics. It will

Table 6.3: Workload logs

Logfile (ω)	Collection period	Total Jobs	Total hours	Total no. of Procs. (M_ω)	Jobs / hour (J_ω)	Avg. runtime in hour (T_ω)	Avg procs. per job (P_ω)	Normalized load (η_ω)
DAS2-fs0-2003	12 months	225,711	8744	144	25.81	1.09	10.27	2.01
DAS2-fs1-2003	12 months	40,315	8633	64	4.67	1.23	8.38	0.75
DAS2-fs2-2003	12 months	66,429	8760	64	7.58	1.29	9.45	1.44
DAS2-fs3-2003	12 months	66,737	8712	64	7.66	1.17	4.96	0.69
DAS2-fs4-2003	11 months	33,795	7963	64	4.24	1.67	3.66	0.40
LLNL-uBGL-2006	7 months	112,611	5339	2,048	21.09	1.25	575.79	7.41
LPC-EGEE-2004	9 months	234,889	5728	140	41.00	1.80	1	0.53

also show us the merits and demerits of going online with VM allocation mechanisms in clouds.

The input data we use are collected from The Parallel Workload Archive [23], as in the previous chapter. We briefly describe the Parallel Workload Archive here for the sake of completeness. It is a rich collection of well studied and standardized workloads from various grid and supercomputing sites. These workloads provide us with an opportunity to experiment with real data in the absence of publicly available cloud workload data (to the best of our knowledge, there were no such workloads at the time of writing this chapter). In the previous chapter, we used eleven workload logs from the Parallel Workload Archive to compare CA-PROVISION with CA-GREEDY. Here we narrow it down to the seven logs for which CA-PROVISION outperformed CA-GREEDY in all categories. These logs are: 1) DAS2-fs0-2003 - DAS2-fs4-2003, from a research grid of five clusters at the Advanced School of Computing and Imaging in the Netherlands; 2) LLNL-uBGL-2006, from a Blue Gene/L system at Lawrence Livermore National Lab; and 3) LPC-EGEE-2004, from a Linux cluster at The Laboratory for Corpuscular Physics, Univ. Blaise-Pascal, France.

We show the information about the workload logs and some statistics associated with them in Table 6.3. The table shows the name of workload logs, the duration for which the logs were collected, the number of jobs, and the total log hours in the first four columns. The table also shows the total number of processors of the system the logs were generated from. We assume that the weight of a VM instance corresponds to the number of processors allocated to it. Hence, the total number of processors of a system represents the total weight of the computing resources M. We denote the total computing resources allocated to a

Table 6.4: Simulation Parameters

Parameter	Notation	Values
Average bundle weight	s_{avg}	from workload (P_ω)
Average time per job	l_{avg}	from workload (T_ω)
Average deadline factor	d_{avg}	3
Average valuation	v_{avg}	5, 10, 20
Cost of running and idle VM	c_R, c_I	1, 0.5

workload ω by M_ω. The next four columns in Table 6.3 are statistical data and are defined as follows. Jobs/hour (J_ω) is the average number of jobs submitted to the system per hour. The average runtime (T_ω) and the average number of processors per job (P_ω) are calculated over all records of the workload log. The last column shows η_ω, the 'normalized load' of workload ω, which is computed as follows

$$\eta_\omega = \frac{J_\omega \times T_\omega \times P_\omega}{M_\omega}.$$

Normalized load measures the average amount of resources requested against each unit of computing resources available. It helps us in ranking the otherwise heterogeneous workloads and also explaining some of the experimental results.

We setup the simulation experiments as follows. We assume that a cloud provider provisions its resources into four types of VM instances, $VM_1, \ldots, VM_4$ with weights $\mathbf{w} = (1, 2, 4, 8)$. User's bids ($\theta_j = r_1^j, \ldots, r_m^j, a_j, l_j, d_j, v_j$) are generated using a workload file ω and its pre-computed statistics that are shown above. The time of arrival a_j of user j is taken from the log file. $s_j = \sum_{i=1}^{m} w_i r_i^j$, i.e., the total weight of the requested bundle is chosen from an exponential distribution with mean $s_{avg} = P_\omega$, the average number of processors per job in workload ω. We arbitrarily set the values of r_i^js once we determine s_j. The required time for allocation, l_j, is also determined using exponential distribution with mean l_{avg}, which is derived from T_ω, average runtime of the jobs in workload ω. Deadline d_j and valuation v_j are computed as $d_j = a_j + l_j \times \exp(d_{avg})$ and $v_j = \exp(v_{avg})$. We show the values chosen for d_{avg} and v_{avg} in Table 6.4. We bring more variation in the input data

by randomly choosing about 50% users and multiplying any of their bid parameters by 2.

CA-PROVISION computes a reserve price $v_{res} = c_R - c_I$ from the cost parameters associated with a running or idling unit VM instance. Although we did not present MOVMPA with a reserve price, it can be easily incorporated by discarding users below the reserve price and then charging a winning user the reserve price instead of zero at line 20 of Algorithm 9. In our experiments, the reserve price for MOVMPA is the same as that used for CA-PROVISION. We show all the parameter values in Table 6.4. We run 18 experiments per workload log with different combinations of parameters.

6.6.2 Analysis of Results

We summarize the results per workload in Figures 6.1 to 6.3. In this figure, we show the workload logs with their normalized load on the horizontal axis and the percent of users served, the average revenue per served user, and the average utility per served user on the vertical axis. In Figure 6.1, we observe that MOVMPA serves a higher percentage of users than CA-PROVISION. The most significant gain is with logs LLNL-uBGL-2006 (normalized load 7.41) and DAS2-fs0-2003 (normalized load 2.01), which are the highest among the workloads used here. The percentage of served users is nearly doubled for the case of DAS2-fs0-2003 workload and it increases more than six-fold for LPC-EGEE-2004 workload. This is because in CA-PROVISION, even if there are available resources, a user must wait until the next auction time. By that time more users may arrive and she may lose the auction. On the other hand, preemption also makes some users leave even if they receive the allocation in some auction but still require the resources for additional time to complete their task. Since the above two workloads generate high bid density, this leads to an increase in the number of auctions and MOVMPA could accommodate more bids, thanks to its online design.

On the other hand, in Figure 6.2, we see a decline in the average revenue generated from each served user. As we discussed in Section 6.5, online mechanisms cannot obtain an optimal outcome because they determine the allocation based on incomplete information.

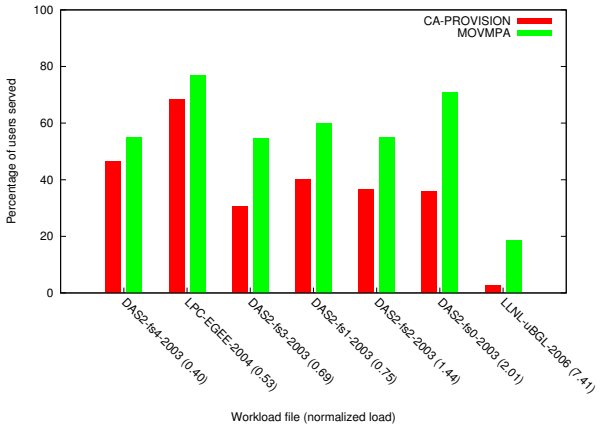


Figure 6.1: Overall results comparing CA-PROVISION and MOVMPA: percent of users served vs. workload logs

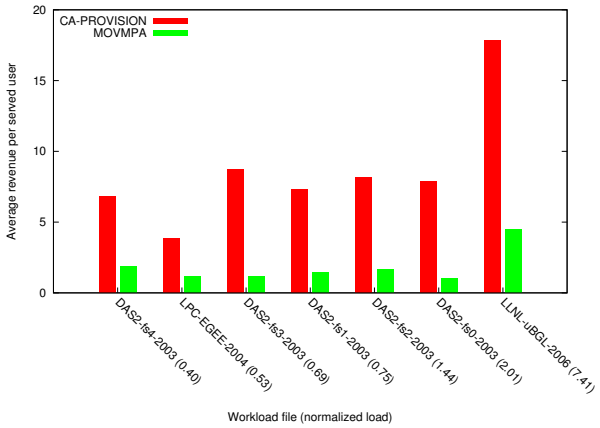


Figure 6.2: Overall results comparing CA-PROVISION and MOVMPA: average revenue per served user vs. workload logs

This leads to a suboptimal value of the social welfare given by Equation (6.1). An intuitive explanation is that by deciding about an allocation as soon as possible, MOVMPA helps a user avoid facing competition with the future bidders. With CA-PROVISION, users must compete with other users that bid during the same period and hence the price of the items increases. But the increased number of served users offsets some of this revenue loss.

However, we see in Figure 6.3 that MOVMPA produces comparable results for the average utility of the served users. Although it is expected that by paying less, users would gain higher utility, but that will happen if both auctions select the same set of users for

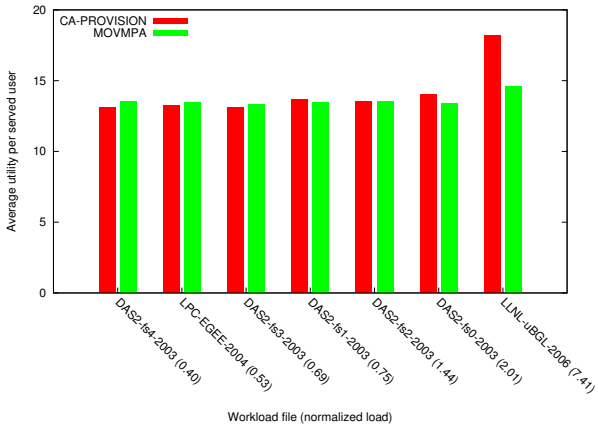


Figure 6.3: Overall results comparing CA-PROVISION and MOVMPA: average utility per served user vs. workload logs

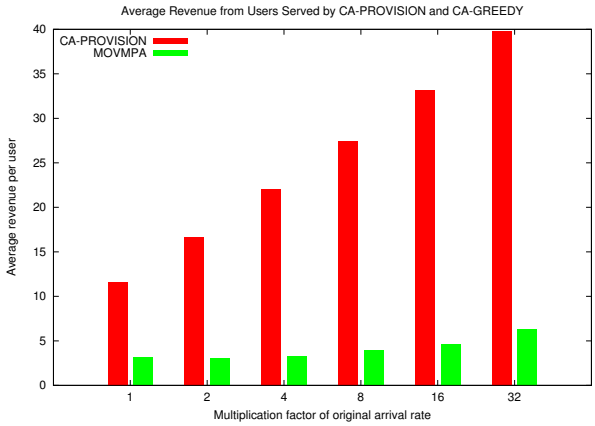


Figure 6.4: Average revenue vs. rate of arrival

allocation. This is not possible because of two reasons: CA-PROVISION preempts a user with low valuation for one with higher valuation, but MOVMPA allocates VMs to a user for the entire period she requested for them. This may lead to losing a high valued user who arrived at a time when there is no resources available. On the other hand, MOVMPA allocates more users than CA-PROVISION, which means it accommodates users with low valuations and these users contribute to the low average utility.

Next, we try in a different way to find out what factors might affect the mechanisms in generating higher revenue for the cloud provider. To do this, we select one log, LPC-EGEE-2004, and tweak its parameters in two different dimensions. In Figure 6.4, we show

Figure 6.5: Average revenue vs. average length of each request

the results obtained by multiplying the arrival rate of the log by a factor of 2, 4, 8, 16, and 32. We see that CA-PROVISION follows the trend of the rate of arrival and generates higher revenue as the rate increases. However, the average revenue generated by MOVMPA increases very slowly. On the other hand, Figure 6.5 shows the average revenue by both mechanisms where the average length of requests (l_{avg}) is multiplied with different factors. We see that this affects MOVMPA more than CA-PROVISION. Since the payment in MOVMPA is determined by finding the critical value in an interval, longer allocation requests increase competition among bids submitted about the same time. But only increasing the rate of arrival does not increase the competition in MOVMPA, because it only needs one moment with available resources for a user to get the allocation in low price.

In summary, we claim that MOVMPA improves the overall cloud experience for both the users and the providers. In current auction mechanisms in cloud, the flexibility of price comes with a risk of preemption of the resources. But MOVMPA ensures that the auction environment will suit more to the real computing tasks and still the users will have an option to bid for their required resources. The cloud provider benefits by serving more users, gaining user satisfaction, and eventually increasing its overall revenue.

6.7 Summary

In this research work, we designed an online mechanism for VM provisioning and allocation in clouds. The mechanism provisions and allocates VM instances whenever enough resources and matching bids are available. We proved that the mechanism is incentive compatible, has a competitive ratio of M and runs in polynomial time. We performed extensive experiments to determine the strength and weaknesses of the mechanism. The mechanism increases the number of users served but at the cost of decreased average revenue. However, the loss in average revenue may be offset by the increased number of served users. We conclude that the proposed mechanism is a good choice for provisioning and allocation of VM instances in clouds.

CHAPTER 7: FUTURE RESEARCH DIRECTIONS

In this chapter, we outline possible research projects that may advance our research in different directions in the future. We believe that our research will encourage new research work in the area of resource provisioning and allocation in clouds. Here we mention three research directions that can be pursued following our work.

7.1 Combinatorial Auction-Based Mechanisms

A direct extension of our work would be to setup a private cloud and implement the mechanisms. These mechanisms then may be tested with simulated and actual workloads. A theoretical extension of our work would be to consider a cloud computing platform that does not have any preset configuration for virtual machines. In this setting, the users will specify the resources they need on their VM instances and the mechanism will provision the resources dynamically for them, following a combinatorial auction. Currently, the cloud computing platforms offer VM instances with fixed configurations. Therefore, this research can be a part of a project aiming at a more flexible future cloud provisioning and allocation system.

7.2 Bidding in Combinatorial Auction-Based Mechanisms

We designed an efficient bidding strategy for submitting non-malleable parallel jobs in clouds. This research can be extended towards designing bidding strategies for different types of jobs, eventually achieving a general bidding strategy that includes many different

types of tasks to be submitted on clouds. There are also open problems in assessing the effect of the presence of different types of users (e.g., risk neutral, risk averse) in the system. This research will be more interesting once cloud providers start deploying combinatorial auction-based mechanisms to actually allocate VM instances through them.

7.3 Bidding Languages for Combinatorial Auctions in Clouds

We investigated the problem of combinatorial auction-based mechanism design for single-minded users. Although we provided insight about how different user behaviors can be modeled using the single-minded property, it would be interesting to design mechanisms for users with multiple alternatives in mind. In that case, researchers will be required to define new bidding languages for VM allocation in clouds and design mechanisms pertaining to these languages. The work by Wang *et al.* [66] could be a good starting point for research in this direction.

7.4 Auction-Based Marketplace for Federation of Clouds

Recently, researchers investigated the idea of building federations of clouds [10], but to the best of our knowledge, it is not deployed as a real system yet. However, building federations of clouds will open a new track of research problems in auction-based mechanisms. We envision that a combinatorial auction-based marketplace would be the center of resource allocation in a federated cloud. In such a marketplace, both users and providers will have an opportunity to host and take part in auctions. This marketplace may also implement an exchange for allocating cloud resources.

CHAPTER 8: CONCLUSION

In this Ph.D. dissertation, we presented our research accomplishments in the field of combinatorial auction-based virtual machine provisioning and allocation in clouds. We discussed the background knowledge and presented a survey of relevant literature to lay the foundation of our work. We identified important open problems in the field, formalized them, and solved the problems by designing mechanisms and algorithms for them. We evaluated each mechanism, both theoretically and experimentally. We discussed the advantages and limitations of our solutions and identified their application areas. Throughout our thesis, we focused on a specific problem domain and investigated many different aspects of this problem to get a comprehensive picture of the domain. Finally, we outlined future directions of research that may stem from our work.

We believe that our thesis is a significant contribution to both the cloud computing industry and academic research. Following its many successful implementations in different fields, we adapted combinatorial auction-based mechanisms to solve the VM provisioning and allocation problem in clouds. These mechanisms will provide the cloud providers the flexibility of dynamically determining the price of their resources. Also, the providers will be free from building complex pricing models or generating user statistics for prediction of system usage. On the other hand, different types of users will be able to select their convenient and economic usage of cloud resources. If a huge system like cloud computing platforms start using combinatorial auctions, it will drive more theoretical researchers to further improve the efficiency of these mechanisms. Finally, yet another successful implementation of combinatorial auctions in computing will encourage computer science researchers to look into other problems that may be solved using combinatorial auctions or auction-based mechanisms in general.

REFERENCES

[1] J. Altmann, C. Courcoubetis, G. D. Stamoulis, M. Dramitinos, T. Rayna, M. Risch, and C. Bannink. GridEcon: A market place for computing resources. In *Proc. Workshop on Grid Economics and Business Models*, pages 185–196, 2008.

[2] Amazon. Amazon EC2 pricing. `http://aws.amazon.com/ec2/pricing/`.

[3] Amazon. Amazon EC2 spot instances. `http://aws.amazon.com/ec2/spot-instances/`.

[4] Amazon. Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2/`.

[5] N. An, W. Elmaghraby, and P. Keskinocak. Bidding strategies and their impact on revenues in combinatorial auctions. *J. Revenue and Pricing Manag.*, 3(4):337–357, 2005.

[6] A. Andersson, M. Tenhunen, and F. Ygge. Integer programming for combinatorial auction winner determination. In *Proc. Fourth International Conference on Multi-Agent Systems*, pages 39–46, 2000.

[7] A. Archer, C. Papadimitriou, K. Talwar, and É. Tardos. An approximate truthful mechanism for combinatorial auctions with single parameter agents. *Internet Mathematics*, 1(2):129–150, 2005.

[8] A. Archer and É. Tardos. Truthful mechanisms for one-parameter agents. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science*, pages 482–491, 2001.

[9] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. In *Proc. 3rd IEEE Int'l Conf. on Cloud Computing Technology and Science*, 2011.

[10] R. Buyya, R. Ranjan, and R. N. Calheiros. InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proc. 10th International Conference on Algorithms and Architectures for Parallel Processing*, pages 13–31, 2010.

[11] E. Campos-Nanez, N. Fabra, and A. Garcia. Dynamic auctions for on-demand services. *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans*, 37(6):878–886, 2007.

[12] T. E. Carroll and D. Grosu. Incentive-compatible online scheduling of malleable parallel jobs with individual deadlines. In *Proc. 39th Intl. Conf. on Parallel Processing*, pages 418–425, 2010.

[13] F. Chang, J. Ren, and R. Viswanathan. Optimal resource allocation in clouds. In *Proc. 3rd IEEE Intl. Conf. on Cloud Computing*, pages 418–425, 2010.

[14] J. Chen, X. Chen, and X. Song. Bidder's strategy under group-buying auction on the internet. *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans*, 32(6):680–690, 2002.

[15] W. Chen, X. Qiao, J. Wei, and T. Huang. A profit-aware virtual machine deployment optimization framework for cloud platform providers. In *Proc. 5th IEEE International Conference on Cloud Computing*, pages 17–24, 2012.

[16] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz. See spot run: Using spot instances for MapReduce workflows. In *Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing*, 2010.

[17] P. Cramton, Y. Shoham, and R. Steinberg. *Combinatorial Auctions*. The MIT Press, 2005.

[18] A. Das and D. Grosu. Combinatorial auction-based protocols for resource allocation in grids. In *Proc. 19th International Parallel and Distributed Processing Symposium, 6th Workshop on Parallel and Distributed Scientific and Engineering Computing*, 2005.

[19] R. K. Dash, P. Vytelingum, A. Rogers, E. David, and N. R. Jennings. Market-based task allocation mechanisms for limited-capacity suppliers. *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans*, 37(3):391–405, 2007.

[20] S. de Vries and R. V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, 15(3):284–309, 2003.

[21] T. Dornemann, E. Juhnke, and B. Freisleben. On-demand resource provisioning for BPEL workflows using amazon's elastic compute cloud. In *Proc. 9th IEEE/ACM Intl. Symp. on Cluster Comp. and the Grid*, May 2009.

[22] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *The American Economic Review*, 97(1):242–259, 2007.

[23] D. G. Feitelson. Parallel Workloads Archives: Logs. `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`.

[24] D. G. Feitelson. Parallel Workloads Archives: Standard Workload Format. `http://www.cs.huji.ac.il/labs/parallel/workload/swf.html`.

[25] D. G. Feitelson. Job scheduling in multiprogrammed parallel systems. Research Report RC 19790 (87657), IBM, 1994.

[26] R. A. Gagliano, M. D. Fraser, and M. E. Schaefer. Auction allocation of computing resources. *Communications of the ACM*, 38(6):88–102, 1995.

[27] S. K. Garg, S. Venugopal, J. Broberg, and R. Buyya. Double auction-inspired meta-scheduling of parallel applications on global grids. *Journal of Parallel and Distributed Computing (in press)*, 2013.

[28] R. Ghosh and V. K. Naik. Biting off safely more than you can chew: Predictive analytics for resource over-commit in IaaS cloud. In *Proc. 5th IEEE International Conference on Cloud Computing*, pages 25–32, 2012.

[29] J. Gomoluch and M. Schroeder. Market-based resource allocation for grid computing: A model and simulation. In *Proc. 1st International Workshop on Middleware for Grid Computing*, pages 211–218, 2003.

[30] D. Grosu. AGORA: An architecture for strategyproof computing in grids. In *Proc. 3rd International Symposium on Parallel and Distributed Computing*, pages 217–224, 2004.

[31] M. Hajiaghayi, R. Kleinberg, and T. Sandholm. Automated online mechanism design and prophet inequalities. In *Proc. National Conference on Artificial Intelligence*, 2007.

[32] J. T. Havill and W. Mao. Competitive online scheduling of perfectly malleable jobs with setup times. *European J. of Oper. Res.*, 187(3):1126–1142, 2008.

[33] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. In *Proc. 1st ACM Symposium on Cloud computing*, 2010.

[34] U. Lampe, M. Siebenhaar, A. Papageorgiou, D. Schuller, and R. Steinmetz. Maximizing cloud provider profit from equilibrium price auctions. In *Proc. 5th IEEE International Conference on Cloud Computing*, pages 38–90, 2012.

[35] D. Lehmann, L. I. O'Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):577–602, 2002.

[36] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Shopping for a cloud made easy. In *Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing*, 2010.

[37] W.-Y. Lin, G.-Y. Lin, and H.-Y. Wei. Dynamic auction mechanism for cloud resource allocation. In *Proc. 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 591–592, 2010.

[38] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *Proc. 7th international conference on Autonomic computing*, 2010.

[39] A. Menychtas, A. Gatzioura, and T. Varvarigou. A business resolution engine for cloud marketplaces. In *Proc. 3rd IEEE Intl. Conf. on Cloud Computing Technology and Science*, pages 462–469, 2011.

[40] Microsoft. Purchase options - pricing - Windows Azure. `http://www.microsoft.com/windowsazure/offers/`.

[41] Microsoft. Windows Azure FAQ. `http://www.microsoft.com/windowsazure/faq/`.

[42] Microsoft. Windows Azure platform, `http://www.microsoft.com/windowsazure/`.

[43] A. Mu'alem and N. Nisan. Truthful approximation mechanisms for restricted combinatorial auctions. In *Proc. 18th National Conference on Artificial Intelligence*, pages 379–384, 2002.

[44] N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[45] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Proc. of the 9th IEEE/ACM Intl. Symp. on Cluster Comp. and the Grid*, pages 124–131, May 2009.

[46] A.-M. Oprescu and T. Kielmann. Bag-of-tasks scheduling under budget constraints. In *Proc. 2nd IEEE Intl. Conf. on Cloud Computing Technology and Science*, pages 351–359, 2010.

[47] D. C. Parkes. Online mechanisms. In N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, chapter 16. Cambridge University Press, 2007.

[48] D. C. Parkes and S. Singh. An mdp-based approach to online mechanism design. In *Proc. 17th Annual Conference on Neural Information Processing Systems*, 2003.

[49] D. C. Parkes, S. P. Singh, and D. Yanovsky. Approximately efficient online mechanism design. In *Proc. 18th Annual Conference on Neural Information Processing Systems*, 2004.

[50] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma. Towards autonomic workload provisioning for enterprise grids and clouds. In *Proc. 10th IEEE/ACM International Conference on Grid Computing*, pages 50–57, 2009.

[51] Rackspace Hosting. `http://www.rackspace.com/`.

[52] M. Risch, J. Altmann, L. Guo, A. Fleming, and C. Courcoubetis. The GridEcon platform: A business scenario testbed for commercial cloud services. In *Proc. Workshop on Grid Economics and Business Models*, pages 46–59, 2009.

[53] M. H. Rothkopf, A. Pekec, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[54] Salesforce. `http://www.salesforce.com/`.

[55] T. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

[56] M. Schwind, T. Stockheim, and O. Gujo. Agent's bidding strategies in a combinatorial auction controlled grid environment. In *Proc. of AAMAS Agent Theories, Architectures, and Languages*, pages 149–163, 2006.

[57] W. Shi and B. Hong. Resource allocation with a budget constraint for computing independent tasks in the cloud. In *Proc. 2nd IEEE Intl. Conf. on Cloud Computing Technology and Science*, pages 327–334, 2010.

[58] P. Shivam, A. Demberel, P. Gunda, D. Irwin, L. Grit, A. Yumerefendi, S. Babu, and J. Chase. Automated and on-demand provisioning of virtual machines for database applications. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 1079–1081, 2007.

[59] X. Sui and H.-F. Leung. An adaptive bidding strategy for combinatorial auction-based resource allocation in dynamic markets. In *Proc. 11th Pacific Rim Intl. Conf. on Artificial Intelligence*, pages 510–522, 2010.

[60] I. E. Sutherland. A futures market in computer time. *Communications of the ACM*, 11(6):449–451, 1968.

[61] W.-T. Tsai and G. Qi. DICB: Dynamic intelligent customizable benign pricing strategy for cloud computing. In *Proc. 5th IEEE International Conference on Cloud Computing*, pages 654–661, 2012.

[62] H. N. Van, F. D. Tran, and J.-M. Menaud. Autonomic virtual resource management for service hosting platforms. In *Proc. ICSE Workshop on Software Engineering Challenges in Cloud Computing*, 2009.

[63] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya. Deadline-driven provisioning of resources fro scientific applications in hybrid clouds with aneka. *Future Generation Computer Systems*, 28:58–65, 2012.

[64] E. Walker, W. Brisken, and J. Romney. To lease or not to lease from storage clouds. *IEEE Computer*, 43(4):44–50, 2010.

[65] C. Wang, H. H. Ghenniwa, and W. Shen. Constraint-based winner determination for auction-based scheduling. *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans*, 39(3):609–618, 2009.

[66] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou. Distributed systems meet economics: Pricing in the cloud. In *Proc. 2nd USENIX Workshop on Hot Topics in Cloud Computing*, 2010.

[67] R. Wang. Auctions versus posted-price selling. *The American Economic Review*, 83(4):838–851, 1993.

[68] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *Intl. J. of High Performance Comp Appl.*, 15(3):258–281, 2001.

[69] S. Yi, D. Kondo, and A. Andrzejak. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Proc. 3rd IEEE Intl. Conf. on Cloud Computing*, pages 236–243, 2010.

[70] M. Zafer, Y. Song, and K.-W. Lee. Optimal bids for spot VMs in a cloud for deadline constrained jobs. In *Proc. 5th IEEE International Conference on Cloud Computing*, pages 75–82, 2012.

[71] S. Zaman and D. Grosu. Combinatorial auction-based allocation of virtual machine instances in clouds. In *Proc. 2nd IEEE Intl. Conf. on Cloud Computing Technology and Science*, pages 127–134, 2010.

[72] S. Zaman and D. Grosu. Combinatorial auction-based dynamic VM provisioning and allocation in clouds. In *Proc. 3rd IEEE Intl. Conf. on Cloud Computing Technology and Science*, pages 107–114, 2011.

[73] S. Zaman and D. Grosu. Efficient bidding for virtual machine instances in clouds. In *Proc. 4th IEEE International Conference on Cloud Computing*, 2011.

[74] S. Zaman and D. Grosu. An online mechanism for dynamic VM provisioning and allocation in clouds. In *Proc. 5th IEEE International Conference on Cloud Computing*, pages 253–260, 2012.

[75] E. Zurel and N. Nisan. An efficient approximate allocation algorithm for combinatorial auctions. In *Proc. 3rd ACM Conference on Electronic Commerce*, pages 125–136, 2001.

ABSTRACT

COMBINATORIAL AUCTION-BASED VIRTUAL MACHINE PROVISIONING AND ALLOCATION IN CLOUDS

by

SHARRUKH ZAMAN

May 2013

Advisor: Dr. Daniel Grosu

Major: Computer Science

Degree: Doctor of Philosophy

Current cloud providers use fixed-price based mechanisms to allocate Virtual Machine (VM) instances to their users. But economic theory states that when there are large amount of resources to be allocated to large number of users, auctions are the most efficient allocation mechanisms. Auctions achieve efficiency of allocation and also maximize the providers' revenue, which a fixed-price based mechanism is unable to do. We argue that combinatorial auctions are best suited for the problem of VM provisioning and allocation in clouds, since they provide the users with the most flexible way to express their requirements. In combinatorial auctions, users bid for bundles of items rather than individual ones, therefore they are able to express whether the items they require are complementary to each other. The *objective* of this Ph.D. dissertation is to design, study, and implement combinatorial auction-based mechanisms for efficient provisioning and allocation of VM instances in clouds. The central hypothesis is that allocation efficiency and revenue maximization can be obtained by inducing users to fully express and truthfully report their preferences to the system. The *rationale* for our research is that, once efficient resource provisioning and allocation mechanisms that take into account the incentives of the users and cloud providers are developed and implemented, it will become more efficient to utilize cloud computing environments for solving challenging problems in business, science and engineering.

In this dissertation, we present three combinatorial auction-based offline mechanisms for provisioning and allocating VM instances in clouds. We also present an online mechanism for dynamic provisioning of virtual machine instances in clouds. Finally, we designed an efficient bidding algorithm to assist users submitting bids to combinatorial auction-based mechanisms to execute parallel jobs on the cloud. We outline our contributions and possible direction for future research in this field.

AUTOBIOGRAPHICAL STATEMENT

Sharrukh Zaman was born in the beautiful South Asian country, Bangladesh. His high school was Mirzapur Cadet College and he completed his undergraduate degree from Bangladesh University of Engineering and Technology (BUET), in Computer Science and Engineering. He worked for about five years in Bangladesh in different positions – lecturer at undergraduate university, software engineer at companies with domestic and offshore clients, and mid-level management personnel/software development lead for a cellphone operator.

Sharrukh started his Ph.D. program in the Department of Computer Science, Wayne State University, in Fall 2007. He joined his advisor Dr. Daniel Grosu's group in Fall 2008 to perform research on application of game theory and mechanism design in distributed systems. His first area of work was replication in distributed systems. In this area, he designed a distributed algorithm to solve the replica placement problem. His next research was on combinatorial auction-based mechanisms in clouds, which eventually became his Ph.D. dissertation topic. He completed a number of research projects in this area and co-authored several papers with his advisor. The papers were published in top-tier conferences in the field, such as the IEEE CloudCom, the IEEE CLOUD, and the CCGrid. He has a few journal articles under review for the IEEE Transactions on Parallel and Distributed Systems and the Journal of Parallel and Distributed Computing.

Following graduation, Sharrukh will join Epic Systems, Verona, Wisconsin as a software developer. Epic is a software provider for healthcare systems and at the time of writing this text, holds about 40% of the market share in this segment.